Christian-Albrechts-Universität zu Kiel

Diplomarbeit

Implementierung eines Statechart-Editors mit layoutbasierten Bearbeitungshilfen

cand. inform. Florian Lüpke

13. Juni 2005

Institut für Informatik und Praktische Mathematik
Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme

Prof. Dr. Reinhard von Hanxleden

betreut durch:
Steffen H. Prochnow

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Kiel,

Zusammenfassung

Im Projekt *Kiel Integrated Environment for Layout (KIEL)* wird ein Werkzeug zur Darstellung, Erstellung, Veränderung und Simulation von *Statecharts* entwickelt. Besonderes Augenmerk wird im *KIEL-Projekt* auf automatisiert angewendetes Layout und innovative benutzerfreundliche Darstellungsformen und Techniken bei der Simulation und Bearbeitung von *Statecharts* gelegt.

Die vorliegende Diplomarbeit beschäftigt sich mit dem Design und der Implementierung eines *Statechart*-Editors als Teil von *KIEL*. Hierfür werden die am Markt verfügbaren *Graph-Editing-Frameworks* dahingehend untersucht, in wieweit sie für die Implementierung eingesetzt werden können. Auf Basis des Rahmenwerks *JGraph* werden dann neben grundlegenden Bearbeitungsfunktionen innovative und bisher in diesem Kontext nicht angewendete Bearbeitungshilfen implementiert und evaluiert. Hierzu zählen die gleitende Strukturveränderung bei automatisiert erfolgenden Layoutmaßnahmen und Platzierungsmechanismen, die Anzeige und das visuelle Abspielen der Bearbeitungsschritt-Historie, mehrschichtige transparente Übersichtsdarstellungen und die farbliche Syntaxprüfung. Dabei sollen sich die Anzeigemodalitäten des Editors an denen von *Esterel-Studio* orientieren in Bezug auf die Darstellung der Diagramm-Elemente. Daneben werden wohlbekannte Funktionen wie das Kollabieren und Expandieren von Zuständen, Diagrammelementtyp-Morphing, rasterorientiertes Bearbeiten, automatische Layouts und syntaxgerichtetes Editieren auf hoher Abstraktionsebene implementiert. Neben diesen funktionalen Merkmalen wird in besonderem Maße die Realisierung einer ergonomischen Benutzeroberfläche, einer intuitiven und komfortablen Bedienbarkeit, Mehrsprachigkeit und vielseitigen Konfigurierbarkeit der Anwendung angestrebt.

Vorwort

Die vorliegende Diplomarbeit entstand im Rahmen des *KIEL*-Projektes, welches an der Christian-Albrechts-Universität zu Kiel am Lehrstuhl für Echtzeitsysteme und Eingebettete Systeme durchgeführt wird. Herrn Prof. Dr. Reinhard von Hanxleden danke ich für abschließende Hinweise zu dieser Arbeit. Mein besonderer Dank gilt dem Betreuer dieser Arbeit, Herrn Dipl. Inform. Steffen H. Prochnow, der eine sehr produktive und angenehme Projektumgebung geschaffen hat, mir eine optimale technische Unterstützung geboten hat sowie sich in zahlreichen und langen Gesprächen vor allem sehr viel Zeit für die Betreuung genommen hat und Engagement in diese Arbeit investiert und viel Geduld mit mir gehabt hat.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Verzeichnis der Abkürzungen

AWT	*Abstract-Windowing-Toolkit*
GUI	*Graphical-User-Interface*
JVM	*Java-Virtual-Maschine*
kByte	kilo Byte
KIEL	*Kiel Integrated Environment for Layout*
MVC	*Model-View-Controller*-Entwurfsmuster
NEA	Nichtdeterministischer Endlicher Automat
OOAD	Objektorientierte Analyse und Design
SSM	*Safe-State-Machine*
SVG	*Scalable Vector Graphics*
UML	*Unified Modelling Language*
XML	*Extensible Markup Language*

1. Einführung

1.1. Ausgangslage

Das *Kiel Integrated Environment for Layout* Projekt (*KIEL*) [46] entwickelt ein Werkzeug zur Darstellung, Erstellung, Veränderung und Simulation von *Statecharts*. Diese sind eine Diagrammform zur Darstellung des Zustandsverhaltens von eingebetteten Systemen, welche auf ihre Umwelt reagieren. Unter eingebetteten Systemen – auch *Reaktive Systeme* genannt – versteht man Computer, die als solche nicht erkennbar sind, weil sie in technischen oder mechanischen Geräten eingebaut sind. Daneben werden *Statecharts* auch in der Softwareentwicklung im Bereich der objektorientierten Analyse und Design (OOAD) eingesetzt, um das Zustandsverhalten von einzelnen Objekten zu modellieren. *KIEL* gehört zur Gruppe der so genannten *Statechart*-Werkzeuge, zu der auch Softwareprodukte wie *Esterel-Studio* und *Statemate* gezählt werden. Besonderes Augenmerk wird im *KIEL*-Projekt auf automatisiert angewendetes Layout und innovative benutzerfreundliche Darstellungstechniken und Eingabehilfen bei der Simulation und Bearbeitung von *Statecharts* gelegt.

Die *KIEL*-Anwendung beinhaltet neben einem Browser, welcher *Statecharts* anzeigt [50], auch eine Dateischnittstelle, welche es erlaubt, *Statecharts* aus *Esterel-Studio* zu importieren. Später soll im *Esterel-Studio*-Format auch exportiert werden können und es sollen Anbindungen zu weiteren *Statechart*-Werkzeugen hinzukommen.

1.2. Ziel der Arbeit

Die vorliegende Diplomarbeit beschäftigt sich mit dem Entwurf und der Implementierung eines *Statechart*-Editors für die *KIEL*-Anwendung. In diesem sollen *Statecharts* erzeugt und bearbeitet werden können. Neben grundlegenden Bearbeitungsfunktionen sollen innovative und bisher im Kontext der *Statechart*-Bearbeitung nicht angewendete Bearbeitungshilfen implementiert und daraufhin evaluiert werden. Dabei sollen sich die Anzeigemodalitäten des Editors an denen von *Esterel-Studio* in Bezug auf die Darstellung der Diagrammelemente orientieren.

Ziel dieser Arbeit ist die Erstellung einer lauffähigen Software, die eine Evaluation sämtlicher implementierter Funktionalitäten ermöglicht. Hierzu ist als Vorarbeit auf der einen Seite die Analyse am Markt verfügbarer *Statechart*-Editoren notwendig. Der Funktionsumfang dieser Editoren soll eingeteilt werden in Grundfunktionalitäten und darüber hinausgehende Merkmale, durch welche sich der jeweilige

Editor von anderen am Markt etablierten Produkten absetzt. Auf der anderen Seite ist eine Evaluierung verfügbarer *Graph-Editing-Frameworks* durchzuführen, um bei der Implementierung des Editors nicht grundlegende Editor-Mechanismen entwickeln zu müssen.

Der aus dieser Arbeit hervorgehende Editor weist die grundlegenden Bearbeitungsfunktionen anderer *Statechart*-Editoren auf und verwertet zusätzliche Ideen und Arbeiten aus einem Forschungsgebiet dieses Lehrstuhls, deren Anwendbarkeit und Nutzen in der vorliegenden Arbeit nach der Fertigstellung des Editors von Benutzern evaluiert werden sollen. Zusammengefasst soll der zu entwickelnde Editor folgende Merkmale aufweisen, die in Kapitel 4 näher erläutert werden:

- Grundlegende Bearbeitungsmöglichkeiten von *Statechart*-Diagrammen: Diagrammelemente erzeugen, löschen, beschriften, verschieben und größenändern
- Syntaxgerichtetes Editieren von *Statecharts*
- Modellorientiert: Unterscheidung zwischen verschiedenen *Statechart*-Dialekten
- Komposition von Diagrammelementen, deren graphische Repräsentationen in SVG definiert sind
- Anwendung von Layouts auf verschiedene Sichten eines *Statecharts*
- Intelligente Bearbeitungstechniken:
  - Verstecken innerer Zustände
  - Layout
  - Layoutgestützte gleitende Strukturveränderung (engl. *Morphing*)
  - Rasterorientiertes Bearbeiten (engl. *Snapping*)
  - Nachvollziehbarkeit von Bearbeitungsschritten
  - Mehrschichtige Übersichtsdarstellungen (engl. *Layering*)
  - Farbliche Syntaxprüfung

Neben den obigen Merkmalen, welche von *KIEL* gefordert werden, wird das weitere Ziel verfolgt, die Qualität des äußeren Erscheinungsbilds und den Funktionsumfang etablierter *Statechart*-Editoren nachzubilden. Hierfür soll der Editor folgende weitere Funktionsmerkmale aufweisen:

- Ansprechendes Benutzeroberflächendesign nach ergonomischen Richtlinien und Standards
- Bereitstellung eines Anwendungsfensters als Rahmen für die *KIEL*-Komponenten Editor und Browser

- Ausrichtungshilfen für die relative Anordnung von Diagrammelementen
- Layouthilfen zum Einfügen von Diagrammelementen (automatisiertes *Platz-Schaffen*)
- *Undo-and-Redo*
- *Cut-and-Paste*
- Zoom
- Typmorphing von Diagrammelementen
- Mehrsprachigkeit (Englisch / Deutsch)
- ausblendbare Hinweismeldungen für Systemereignisse und aufgetretene Fehler, Hinweise zu Werkzeugleistenelementen (engl. *Tooltips*), optische Unterstützung bei Bearbeitungsschritten durch unterschiedliche Mauszeiger
- Konfigurierbarkeit: Unter anderem Ausblendbarkeit von Beschreibungselementen und Diagrammelement-Beschriftungen, Änderbarkeit von Farben, initialen Diagrammelement-Größen, Zoomfaktor und Morphing-Geschwindigkeit
- Bearbeitungsschritt-Historie
- Verschiedene Graphik-Exportformate
- Druck-Funktionalität
- Hervorhebung selektierter Diagrammelemente in allen Ansichtsschichten zur besseren Orientierung

1.3. Aufbau und Überblick

In der vorliegenden Arbeit werden im Anschluß an dieses Kapitel die theoretischen Grundlagen geschaffen (Kapitel 2), welche zum Verständnis der Zielbeschreibung und des Lösungswegs notwendig sind. Hiernach werden zum einen Software-Produkte – sogenannte Rahmenwerke – vorgestellt, welche als Basis zur Entwicklung eines *Statechart*-Editors verwendet werden können (Abschnitt 3.2). Zum anderen werden am Markt verfügbare *Statechart*-Werkzeuge analysiert, welche ähnliche Funktionsmerkmale bieten wie die in dieser Arbeit entwickelte Software (Abschnitt 3.1). Ziel des Kapitels 3 soll die Vermittlung dessen sein, was Stand der Technik ist und soll somit einen Einblick in andere Lösungsstrategien geben. Anschließend werden die funktionalen Anforderungen an die in dieser Arbeit zu entwickelnde Software vorgestellt (Kapitel 4), bevor dann der Weg zur Lösung der gestellten Aufgaben aufgezeigt wird (Kapitel 5). Dieser Weg umfasst neben der Festlegung auf eine Vorgehensweise (Abschnitt 5.1) auch die Entwicklung des Software-Designs

(Abschnitt 5.2) und seiner Implementierung (Abschnitt 5.3). Abschließend werden in den Schlussbemerkungen die Ergebnisse, die Bewertung der gefundenen Lösung sowie ein Ausblick auf Erweiterungsmöglichkeiten vorgestellt (Kapitel 6).

Im Anhang findet sich neben einem Benutzerhandbuch auch der gesamte Programm-Quelltext des Editors.

2. Theoretische Grundlagen

Nach einer Erläuterung des Begriffs *Reaktive Systeme* in Abschnitt 2.1 werden in diesem Kapitel formale Beschreibungen für solche Systeme vorgestellt. Die Grundlage der Modellierung Reaktiver Systeme bildet der Formalismus der *Statecharts*, welche in Abschnitt 2.3.2 vorgestellt werden. Diese bauen auf den *Endlichen Automaten* auf, welche im Abschnitt 2.2 behandelt werden. Eine Erweiterung der *Statecharts* bilden die *Safe-State-Machines* in Abschnitt 2.3.3. Diese werden in *KIEL* modelliert und sind daher insbesondere interessant für die Implementierung des *KIEL*-Editors. Der Editor baut in seiner Implementierung auf einem so genannten *Graph-Editing-Framework* auf. Daher werden im Abschnitt 2.3.1 Graphen kurz vorgestellt, um als Vorbereitung auf den Abschnitt 3.2 über die Evaluierung von *Graph-Editing-Frameworks* zu dienen. Das Design und die Implementierung des *KIEL*-Editors sollen sich an Vorgehensmodellen der Softwareentwicklung orientieren und Entwurfsmuster einsetzen. Die Grundlagen hierfür werden in Abschnitt 2.4 behandelt.

In den folgenden Abschnitten werden formale Beschreibungen für Systeme vorgestellt, welche diskrete Eingaben, Ausgaben und Zustände besitzen. Jedes solcher Systeme kennt eine nur endliche Menge von Zuständen, und für jede mögliche Eingabe während jedes Zustands ist ein Folgezustand definiert. Ist höchstens ein Folgezustand oder eine Zustandskombination definiert, so spricht man von *deterministischen* Modellen. Hieraus folgt, dass ein Zustand durch bestimmte Eingaben in einer definierten Reihenfolge erreicht wird und jeder Zustand die Reaktion auf folgende Eingaben beschreibt.

Nahezu alle technischen Geräte des täglichen Lebens können als solche Systeme aufgefasst und beschrieben werden. Diese Geräte reagieren in Abhängigkeit ihres aktuellen Zustands auf ihre Umwelt, häufig ist dies ein Mensch. Das Fernsehgerät ist ein Beispiel für ein System mit einer endlichen Anzahl von Zuständen, welches auf Eingaben reagiert und seinen Zustand ändert: In Abhängigkeit von getätigten Eingaben befindet sich das Gerät entweder im Zustand *Normalbetrieb* oder im Zustand *Videotext-Anzeige*. Daneben existieren die möglichen Zustände *Ton eingeschaltet* und *Ton ausgeschaltet*. In Abhängigkeit davon, welche Eingaben als nächste erfolgen, wird z. B. der Kanal oder die Lautstärke geändert oder das Gerät in den Zustand *ausgeschaltet* versetzt.

Formal besitzt dieses System:

- eine endliche Menge von Zuständen (z. B. *Videotext-Anzeige*),
- eine endliche Menge von Zustandsübergängen, ausgelöst durch eine Eingabe (z. B. das Umschalten auf Videotext-Anzeige mit *Videotext*-Button auslösen),

- eine Menge von Aktionen (z. B. *Lautstärke ändern*) und
- einen Startzustand sowie einen Endzustand (z. B. *Fernsehgerät ausgeschaltet*).

Hierbei ist anzumerken, dass das Verhalten, beziehungsweise die Aktionen, unterschiedlich mit Zuständen und Zustandsübergängen assoziiert sein können. In Abschnitt 2.3 wird für diese Zustandsbeschreibung des Systems *Fernsehgerät* eine graphische Notation in Form eines so genannten *Statecharts* vorgestellt. Abbildung 2.3 stellt diese Zustandsbeschreibung in Form eines *Statecharts* dar.

Beispiele für Systeme mit endlichen Zustandsmengen finden sich auch in der Informatik. Die Automatentheorie, ein Teilgebiet der Theoretischen Informatik, beschäftigt sich mit der Untersuchung der Eigenschaften solcher Systeme und entwickelt hierfür passende Modelle, welche Automatenmodelle oder kurz Automaten genannt werden, z. B. den von Mealy und Moore eingeführten Endlichen Automaten und die von Harel eingeführten *Statecharts*. Neben der Beschreibung von Reaktiven Systemen, wie z. B. dem Fernsehgerät, eignet sich die Automatentheorie z. B. auch zur Analyse von Algorithmen und zur Beschreibung von regulären Ausdrücken. Grundsätzlich kann das Modell des Endlichen Automaten zur Lösungsfindung von Problemen verwendet werden, welche sich mittels einer endlichen Zahl von Zuständen darstellen lassen. In solch einer Darstellung wird nach einer Reihe von Eingaben gesucht, welche dazu führen, dass der Automat aus einem Startzustand heraus über Zwischenzustände einen bestimmten gewünschten Endzustand erreicht. Der Vorteil des Einsatzes von Automatenmodellen liegt also darin, dass sie sich auf der einen Seite für den Einsatz in vielfältigen Aufgabenstellungen eignen und auf der anderen Seite schon sehr umfassend erforscht sind, somit die Kenntnis ihrer Charakteristika der Lösungsfindung vielfältiger Problemstellungen dient.

Automatenmodelle, z. B. die Endlichen Automaten, können graphisch in Form von Zustandsdiagrammen dargestellt werden. Eine Form von erweiterten Zustandsdiagrammen sind *Statecharts*. Zustandsdiagramme eignen sich nicht nur für oben genannte Einsatzgebiete sondern auch zur Beschreibung von Softwaresystemen und sind daher auch Teil der *Unified Modelling Language (UML)* [35], siehe Abschnitt 2.4.1. Zustandsdiagramme werden seit einigen Jahren in Softwareentwicklungsprojekten häufig eingesetzt neben anderen *UML*-Diagrammformen wie Klassendiagrammen und Sequenzdiagrammen, auf die hier jedoch nicht näher eingegangen wird. Bei der Modellierung von objektorientierten Softwaresystemen werden Zustandsdiagramme genutzt, um die Folge von Zuständen darzustellen, welche ein Objekt während der Programmlaufzeit einnehmen kann und aufgrund welcher Ereignisse Zustandsänderungen stattfinden [36].

2.1. Reaktive Systeme

Der Begriff der Reaktiven Systeme bezeichnet Systeme, die permanent mit ihrer Umwelt kommunizieren und interagieren. Charakteristisch ist hierbei, dass Reak-

tive Systeme auf Ereignisse aus ihrer Umwelt reagieren, und daher ihr Verhalten in hohem Maße von der Umwelt abhängt. Reaktive Systeme bestehen häufig aus voneinander unabhängig entwickelten Teilsystemen, welche ihrerseits wiederum vielfach Reaktive Systeme sind und untereinander kommunizieren, oder sie lassen sich willkürlich hierarchisch gliedern in Subsysteme. Einen hohen Grad an Nebenläufigkeit weisen Reaktive Systeme häufig auf, da einerseits die Teilsysteme parallel ausgeführt werden und andererseits das Gesamtsystem parallel zu seiner Umgebung abläuft, und die Umgebung wiederum unter Umständen als ein Reaktives System interpretiert werden kann.

Die Steuereinheiten technischer Vorgänge z. B. in Fahrzeugen sind ebenso Reaktive Systeme wie Betriebssysteme und graphische Benutzeroberflächen. Die Anforderungen, welche in diesen Einsatzbereichen gestellt werden bezüglich der Kommunikation und Reaktion des Systems auf seine sich ständig wechselnde Umgebung, werden durch herkömmliche transformationelle Systeme nicht erfüllt. Transformationelle Systeme unterscheiden sich von Reaktiven Systemen darin, dass erstere nur eine begrenzte Laufzeit haben und nur in geringerem Maße mit ihrer Umwelt interagieren.

Der hohe Grad an Kommunikation zwischen den Teilsystemen untereinander und mit der Systemumgebung bedeutet ein erhöhtes Maß an Aufwand bei der Spezifikation und Implementierung solcher Systeme. Daher wurden spezielle Formalismen entwickelt, welche die typischen Eigenschaften dieser Systeme, z. B. Hierarchie, Parallelität und Interaktion, darzustellen vermögen. Zu diesen Formalismen gehören z. B. Prozessalgebren [16], Petrinetze, Ereignisstrukturen sowie graphische Spezifikationssprachen wie *Message-Sequence-Charts*, *Statecharts* und die dynamischen Modelle der *UML* (Abschnitt 2.4.1). Die vorliegende Arbeit beschäftigt sich ausschließlich mit *Statecharts*, diese werden in Abschnitt 2.3.2 vorgestellt.

2.2. Endliche Automaten als Grundlage der Modellierung Reaktiver Systeme

Der Endliche Automat ist eine formale Beschreibung eines Systems, welches sich zu jedem Zeitpunkt in genau einem Zustand befindet, welches insgesamt eine endliche Zahl von Zuständen kennt und welches Eingaben verarbeitet und Ausgaben liefert. Eingaben können externe Ereignisse sein und Ausgaben können z. B. in Textform erfolgen. Das Modell des Endlichen Automaten besteht daher aus einer Menge von Zuständen und einer Zustandsüberführungsfunktion, welche einem Paar, bestehend aus Ereignis, auch Eingabe genannt, und einem Zustand einen neuen Zustand und optional eine Ausgabe zuordnet [22]. Solch eine Funktion beschreibt also das Reaktionsverhalten eines Systems in Form von Zustandsänderungen. Den Übergang von einem Zustand zu einem anderen bezeichnet man als *Zustandsübergang* oder *Transition*. Ein Zustandsübergang tritt als Reaktion auf ein äußeres Ereignis auf. Bei einer Softwareanwendung ist dies z. B. bei einer Benutzeraktion der Fall. Des Weiteren kennt ein Endlicher Automat einen Startzustand und eine Menge

von Endzuständen. Erreicht ein Automat durch geeignete Eingaben einen dieser Endzustände, so sagt man, dieser Automat habe die Eingaben in der erfolgten Reihenfolge *akzeptiert*. Besteht jede Eingabe z. B. aus genau einem Buchstaben, so sagt man, der Automat produziere eine bestimmte Menge von Wörtern, indem er diese Wörter akzeptiert. Auf diese Weise lassen sich unter anderem reguläre Ausdrücke definieren.

Es werden unterschieden Endliche Automaten *ohne* Ausgabe von Endlichen Automaten *mit* Ausgabe. Erstere kennen als einzige Resultate *Eingabe akzeptiert* und *Eingabe nicht akzeptiert*, je nachdem, ob nach Abarbeitung der Eingabe ein Endzustand erreicht wurde oder nicht.

2.3. Grundlagen der Visualisierung Reaktiver Systeme

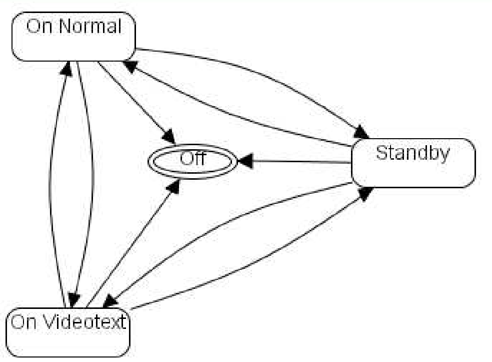


Abbildung 2.1.: Zustände eines TV-Gerätes

Wie im vorhergehenden Abschnitt bereits erwähnt, bilden Endliche Automaten die Grundlage der Modellierung Reaktiver Systeme. Endliche Automaten werden mittels Zustandsdiagrammen visualisiert, welche ihrerseits auf Graphen-Diagrammen aufbauen. Ein Beispiel für ein Zustandsdiagramm zeigt Abbildung 2.1, auf der vereinfacht Zustände eines TV-Gerätes mit ihren Transitionen dargestellt werden.

Es werden zunächst die Graphen-Diagramme eingeführt in Abschnitt 2.3.1, um gleichzeitig die Basis für den Abschnitt über *Graph-Editing-Frameworks* (Abschnitt 3.2) zu schaffen. Hiernach werden in Abschnitt 2.3.2 die *Statecharts* betrachtet, welche auf Graphen und Zustandsdiagrammen aufbauen. Im darauf folgenden Abschnitt 2.3.3 werden die *Safe-State-Machines* beschrieben, welche ihrerseits wiederum *Statecharts* erweitern. Gegenstand dieses Kapitels sind jedoch nicht Lösungen für die Darstellung großer Zustandsmengen wie in [39] beschrieben. Einige solcher Ideen werden in Kapitel 4 angeführt.

2.3.1. Graphen

Graphen stellen eine Generalisierung von Listen- und Baum-Strukturen dar und finden Anwendung, wenn die Relationen zwischen Elementen einer Menge nicht listen- oder baumförmig dargestellt werden können. Beispielsweise hat in einem Baum ein Element höchstens einen Vater sowie beliebig viele Kindelemente, wohingegen man bei einem Graph nur von beliebig vielen Nachbarn spricht. Formal sind Graphen Strukturen, welche aus einer Knotenmenge und einer Kantenmenge bestehen, wobei eine Kante typischerweise zwei Knoten miteinander verbindet, also ein Paar $(\mathcal{V}, \mathcal{E})$ mit $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, wobei die Elemente aus $\mathcal{V}$ Knoten (engl. *Vertexes*) und die Elemente aus $\mathcal{E}$ Kanten (engl. *Edges*) genannt werden [10].

Einsatzgebiete

Mit Graphen lassen sich vielfältige Anwendungen modellieren, z. B. Verkehrsnetze, Moleküle und Flussdiagramme von Computerprogrammen. Eine klassische Aufgabe ist das Finden einer kürzesten Route zwischen zwei Orten. Hierbei wird das Verkehrsnetz als Graph modelliert, dessen Knoten Orte repräsentieren und dessen Kanten mit Gewichten versehen werden, welche die Entfernung zwischen den Knoten darstellen. Die Graphentheorie bietet Algorithmen für solche Graphenprobleme.

Graphische Darstellung

In dieser Arbeit ist nur die graphische Darstellung von Graphen relevant. Die Knoten werden im Nachfolgenden immer als Kreise oder Rechtecke dargestellt, die Kanten als je zwei Knoten verbindende Polygone oder Spline-Kurven. Es werden im Folgenden nur gerichtete Graphen betrachtet werden, deren Kanten im Gegensatz zu ungerichteten Graphen eine durch einen Pfeil markierte Richtung besitzen.

2.3.2. *Statecharts*

Endliche Automaten werden graphisch in Form so genannter Zustandsdiagramme dargestellt, welche ihrerseits als Erweiterung gerichteter Graphen betrachtet werden können. *Statecharts* wiederum erweitern Zustandsdiagramme mit dem Ziel, eingebettete Systeme adäquat darstellen zu können. Ein Beispiel für ein *Statechart*, welches das vereinfachte Zustandsverhalten eines TV-Gerätes modelliert, stellt Abbildung 2.2 dar. *Statecharts* sind ausdrucksstärker als Endliche Automaten, da mit ihnen auch hierarchische Strukturen, Parallelität und Kommunikation modelliert und Pseudozustände definiert werden können, siehe hierzu die folgenden Abschnitte. Das Verhalten eines Reaktiven Systems wird dabei beschrieben als die Menge an erlaubten Eingabeereignis- und Ausgabeereignis-Sequenzen, Bedingungen, Aktionen und Zeitbeschränkungen. Diese Elemente werden ebenfalls in den folgenden Abschnitten vorgestellt.

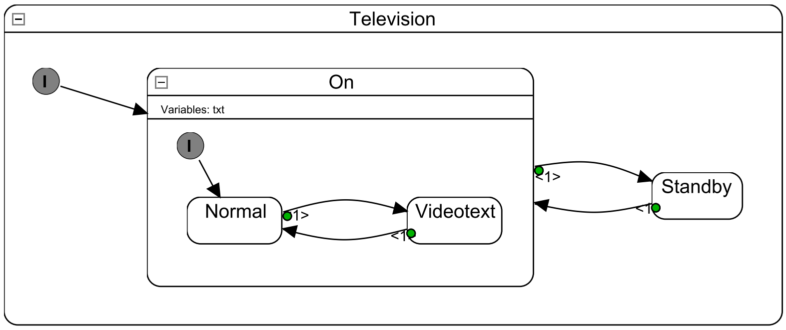


Abbildung 2.2.: Zustände eines TV-Gerätes

Statecharts wurden von David Harel eingeführt, um das komplexe Verhalten Reaktiver Systeme übersichtlich zu beschreiben [19]. Industrielle Werkzeuge, z. B. *Rhapsody*, *Statemate*, *Artisan*, *Matlab/Stateflow* und *Esterel-Studio* bieten ein so genanntes *Forward-Engineering* an, ähnlich dem aus anderen *UML*-Werkzeugen bekannten Verfahren, welches erlaubt, aus *Statechart*-Diagrammen Programm-Quelltext zu erzeugen. Im Laufe der Zeit haben sich allerdings verschiedene *Statechart*-Dialekte entwickelt. In diesem Abschnitt werden die Harel-*Statecharts* vorgestellt. Andere Dialekte sind z. B. die *Safe-State-Machine* – verwendet in *Esterel-Studio* – welche im Abschnitt 2.3.3 beschrieben werden und die *UML-Statecharts*, welche z. B. in *ArgoUML* (Abschnitt 3.1.4) Verwendung finden. Die Entwicklung verschiedener *Statechart*-Dialekte führte dazu, dass die genannten Werkzeuge auf unterschiedlichen Dialekten und damit unterschiedlichen *Statechart*-Semantiken basieren. Daher wird aus dem gleichen *Statechart*-Modell durch unterschiedliche Werkzeuge Programm-Quelltext mit unterschiedlichem Verhalten generiert. Des Weiteren bieten einige dieser Werkzeuge das Simulieren von *Statecharts* an. Hierbei kann der Benutzer die Rolle der Umwelt übernehmen, indem er an der Tastatur bestimmte Ereignisse produziert, und das Werkzeug zeigt mittels farblicher Markierungen, in welchen Zuständen sich das Reaktive System gerade befindet. Jedoch auch in diesem Anwendungsbereich zeigen sich semantische Unterschiede der einzelnen Dialekte. Interessant für die vorliegende Arbeit ist allerdings nur, dass die Syntax und damit die graphische Repräsentation je nach Dialekt und damit je nach Werkzeug verschieden ist.

Es wird im Folgenden nur die Syntax und die graphische Repräsentation der *Statecharts* betrachtet. Die Semantik wird z. B. in [19] beschrieben und es wird hier nur aus Gründen der Motivation und auch nur am Rande auf sie eingegangen.

Graphische Darstellung

Statecharts bestehen aus Zuständen und Transitionen – in Graphen Knoten und Kanten genannt – Transitionsbeschriftungen, Zustandsbeschriftungen, *Event*-Deklarationen und Variablen-Deklarationen. In den folgenden Abschnitten wird beschrieben, wie diese Diagrammelemente in *KIEL* dargestellt werden, da diese Darstellungsform von dem in dieser Arbeit zu entwickelnden Editor übernommen werden soll. Diese Form der graphischen Repräsentation orientiert sich an der *Esterel-Studio*-Notation.

Zustände

Hierarchische Strukturen dienen nicht nur der Übersichtlichkeit und erweitern die Syntax und damit die Semantik von Endlichen Automaten, sondern sie fördern auch eine hierarchische Entwicklung von Softwarekomponenten. Unter Hierarchie wird hier verstanden, dass ein Zustand Unterzustände enthalten kann. Solche Zustände werden *OR-States*, *Macro-States* oder *Processes* genannt oder im Falle von Parallelität *Regions* eines *AND-States*. Wird ein solcher hierarchischer Zustand aktiv, so wird entweder durch darin enthaltenen *Initial-State* oder den *History-State* definiert, welcher Unterzustand aktiv wird. In Abbildung 2.2 sind die mit *On* und *Television* beschrifteten Zustände *OR-State* und die mit *I* beschrifteten *Initial-States*. Darüberhinaus handelt es sich bei den mit *Normal* und *Videotext* beschrifteten Zuständen um sogenannte *Simple-States*.

Sollen zwei oder mehr Zustände auf der gleichen Hierarchiestufe gleichzeitig aktiv sein, so werden sie als Unterzustände einem *AND-State* hinzugefügt und voneinander getrennt durch eine gestrichelte Linie, eine so genannte *Delimiter-Line*. Solche Unterzustände heißen *Regions*. Beispielsweise handelt es sich bei dem in Abbildung 2.3 dargestellten mit *On* beschrifteten Zustand um einen *AND-State*. Ist dieser *AND-State* aktiv, so gibt es in jeder seiner *Regions* einen aktiven Zustand. Durch diese Konstruktion kann die Anzahl an Zuständen eines *Statecharts* insgesamt verringert werden [19].

Zusammenfassend besteht ein *Statechart* aus folgenden Zuständen, welche in den Abbildungen 2.2 und 2.3 dargestellt werden.

- *Simple-State*: Wird dargestellt durch ein Rechteck mit abgerundeten Kanten und kann Beschriftungen enthalten (Abschnitt 2.3.2).
- *OR-State*: Wird dargestellt wie ein *Simple-State*, jedoch mit einer zusätzlichen horizontalen Geraden, welche eine obere Zeile im Rechteck bildet, in der die Beschriftung gezeichnet wird.
- *AND-State*: Wird dargestellt wie ein *OR-State* und enthält gestrichelte waagerechte und horizontale Geraden, welche das Rechteck in Bereiche aufteilen.
- *Region*: Beschreibt einen Bereich eines *AND-States* und besitzt keine graphische Repräsentation.

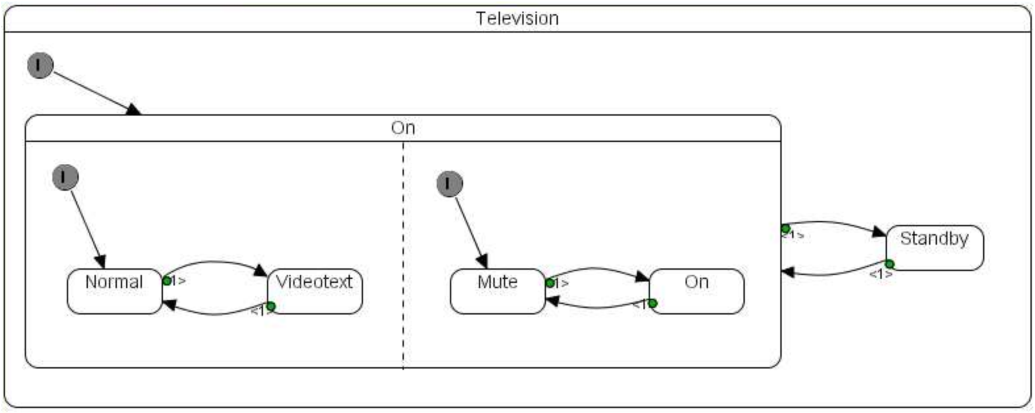


Abbildung 2.3.: Parallelität: *AND-State*

Zustände, welche weitere Zustände enthalten dürfen, – namentlich sind dies *OR-States*, *AND-States* und *Regions* – werden auch *Composite-States*, *Macro-States* oder zusammengesetzte Zustände genannt.

Zustandsbeschriftungen

Die folgenden Zustandsbeschriftungen bestehen aus syntaktischen Elementen, auf deren Bedeutung hier nicht näher eingegangen wird. Dies sind sogenannte *On-Entry-Actions*, *On-Inside-Actions*, *On-Exit-Actions*, *(Local) Events* und *(Local) Variables*, sowie *Statechart*-globale *Input-Events* und *Output-Events*.

- *On-Entry-Actions*, *On-Inside-Actions* und *On-Exit-Actions* werden im Inneren des jeweiligen Zustands und durch waagerechte Linien voneinander getrennt angezeigt, siehe Abbildung 2.4.

- *(Local) Events* und *(Local) Variables* bei *Composite-States* werden unter dem Zustandsnamen im Inneren des Zustands angezeigt, ebenfalls durch Linien voneinander getrennt, siehe Abbildung 2.4.

- *Statechart*-globale *Input-Events* und *Output-Events* werden nur in einem Bearbeitungsdialog angezeigt, der aus einem Menü heraus aufrufbar ist. Diese Elemente sind im *Statechart* nicht sichtbar.

Pseudozustände

Sämtliche *Pseudo-States* werden als Kreis dargestellt mit einem Buchstaben, der den Typ des *Pseudo-States* angibt. Pseudozustände werden sofort wieder verlassen, sobald sie betreten werden.

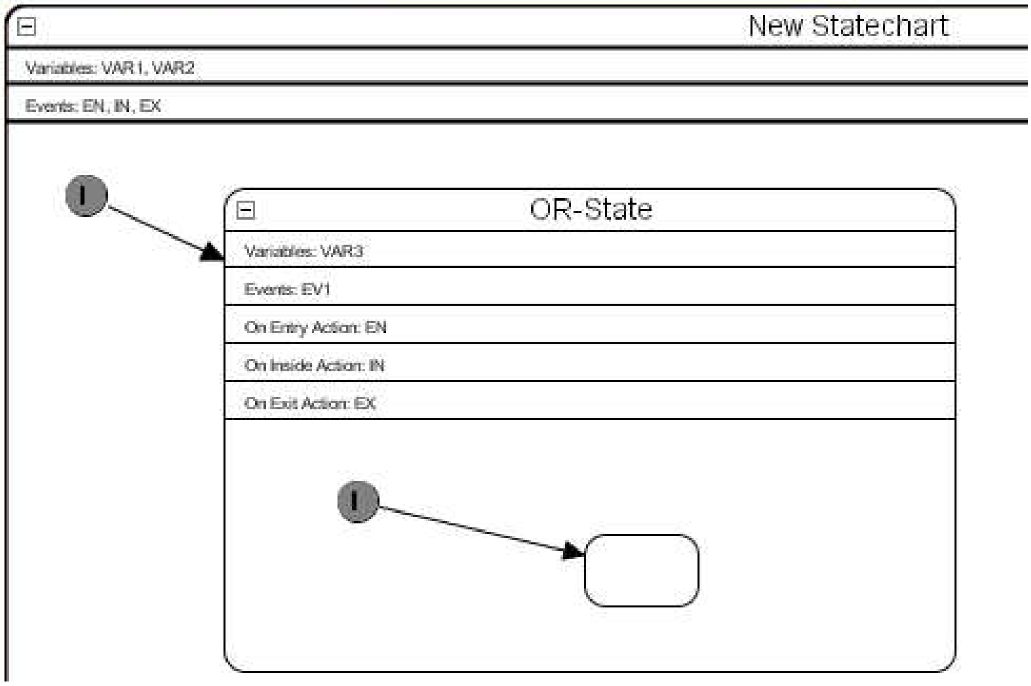


Abbildung 2.4.: Zustandsbeschriftungen

- *Initial-State*: Jeder *OR-State* und jede *Region* besitzen genau einen *Initial-State*. Ein *Initial-State* wird als grau ausgefüllter Kreis mit der Beschriftung 'I' in der Mitte dargestellt.

- *Choice*, auch *Dynamic-Choice* und *Conditional* genannt: Besitzt mehrere eingehende und ausgehende Transitionen, wobei nur die eingehenden Transitionen durch bestimmte Ereignisse aktiviert werden, und die Aktivierung der eingehenden und ausgehenden Transitionen an Bedingungen geknüpft werden kann. Ein *Choice* wird als grau ausgefüllter Kreis mit der Beschriftung 'C' in der Mitte dargestellt.

- *History-State*: *OR-States* und *Regions* können maximal einen *History-State* besitzen, welcher anzeigt, dass bei erneutem Betreten seines Vaters sofort derjenige Zustand aktiv wird, welcher beim Verlassen des Vaters zuletzt aktiv war, sofern der Zustand durch eine auf den *History-State* zeigende Transition betreten wird. Ein *History-State* wird als grau ausgefüllter Kreis mit der Beschriftung 'H' in der Mitte dargestellt.

- *Deep-History-State*: Es gilt das gleiche wie für *History-States*, jedoch mit der Erweiterung, dass bei Betreten des Vaters die zuletzt aktiven Zustände aller Hierarchiestufen wieder aktiv werden und dass ein *Deep-History-State* als Beschriftung ein 'H*' in der Mitte trägt.

Transitionsbeschriftungen

Eine Transition kann eine Beschriftung besitzen, welche die Beschreibung eines sie auslösenden Ereignisses (engl. *Trigger*), einer Bedingung (engl. *Condition*) und einer Aktion (engl. *Action*) beinhalten kann. Dabei sind alle diese drei Elemente optional. Ein *Trigger* bezeichnet die Menge an Ereignissen, welche eintreten müssen, damit eine Transition aktiviert wird. Unter einer *Condition* wird ein logischer Ausdruck verstanden, welcher aus Variablen aufgebaut sein kann und von dem zusätzlich abhängt, ob eine Transition aktiviert wird. Die *Action* bezeichnet eine Menge von Aktionen, welche Variablen ändern und Ereignisse auslösen können und welche ausgeführt werden, sobald die Transition aktiviert wird. Transitionsbeschriftungen von und nach Pseudozuständen unterliegen teilweise gewissen Beschränkungen, welche hier jedoch nicht weiter betrachtet werden sollen.

Transitionen

Transitionen können eine Transitionsbeschriftung besitzen. Durch die Aktivierung einer Transition wird der Quellzustand der Transition inaktiv und der Zielzustand aktiv. Eine Transition kann als Polygon oder Spline-Kurve dargestellt werden und beliebig viele Stützpunkte enthalten.

2.3.3. *Safe-State-Machines*

Mit *Safe-State-Machines (SSM)*, auch bekannt unter der Bezeichnung *SyncCharts*, wird ein visuelles synchrones Automatenmodell bezeichnet [7]. Dieses wurde entworfen als eine graphische Notation für die Sprache *Esterel* und stellt einen Dialekt und gleichzeitig eine echte Obermenge der *Statecharts* dar. In den nachfolgenden Kapiteln ist bei Verwendung des Begriffs *Statechart* der *SSM*-Dialekt gemeint. Nur *diejenigen* syntaktischen Konzepte werden im Folgenden aufgeführt, um welche die *SSM* die *Statecharts* erweitern. Beschrieben wird im Folgenden die Darstellung dieser Elemente in *KIEL*.

Suspendierung

Es kann für das Auftreten bestimmter Ereignisse bestimmt werden, dass alle in einem bestimmten Zustand befindlichen Prozesse solange suspendiert werden, bis diese Ereignisse nicht mehr präsent sind. Dies wird modelliert mit dem so genannten *Suspend*-Diagrammelement und dem Transitionstyp *Suspension*. Diese *Suspension* wird aktiviert, wenn jene Ereignisse eintreten und führt in einen *Suspend*. *Suspends* werden als gelb ausgefüllter Kreis mit der Beschriftung 'S' und *Suspensions* durch einen kleinen gelben Kreis am Anfang der Transition gekennzeichnet.

Terminierung

Safe-State-Machines kennen Endzustände (engl. *Final-States*). Sie markieren Zustände, in welchen ein Subsystem eines Reaktiven Systems endet. Solche Zustände

werden durch einen doppelten Rahmen gekennzeichnet. Haben in einem Zustand alle Subzustände einen Endzustand erreicht, so kann mittels einer Terminierungstransition (engl. *Normal-Termination*) dieser Zustand ohne weitere Eingabeereignisse verlassen werden. Solch eine Terminierungstransition wird durch einen kleinen grünen Kreis am Anfang der Transition gekennzeichnet. In *Esterel-Studio* wird stattdessen ein kleines grünes Dreieck dargestellt.

Starker Abbruch

Ein weiteres Modellierungselement stellen die *Strong-Abortion*-Transitionen dar, welche bei Aktivierung sofort ausgeführt werden und alle anderen laufenden Aktionen abbrechen. Diese Transitionen werden durch einen kleinen roten Kreis am Anfang der Transition gekennzeichnet.

Transitionsprioritäten

Grundsätzlich haben *Strong-Abortions* eine höhere Priorität als *Weak-Abortions*. Sobald von einem Zustand mehrere Transitionen abgehen, werden Transitionen gleichen Typs priorisiert, indem diesen automatisch eine Zahl zugewiesen wird, welche die Reihenfolge bei der Auswertung der *Triggers* bestimmt. Je niedriger diese Zahl ist, desto höher ist die Priorität. Diese Zahl wird am Anfang einer Transition dargestellt. Diese Prioritäten können geändert werden. Doppelte Prioritäten können allerdings nicht vergeben werden.

2.4. Vorgehensmodelle zur Softwareentwicklung und Entwurfsmuster

Nachdem in den vorangegangenen Abschnitten auf die Struktur der vom *KIEL*-Editor dargestellten Diagramme eingegangen wurde, sollen in diesem Abschnitt kurz die Grundlagen für das konzeptionelle projekttechnische Vorgehen bei der Umsetzung der Aufgaben in dieser Arbeit, siehe Kapitel 5, umrissen werden.

Je größer Softwareprojekte sind, je komplexer, verteilter und wichtiger, desto mehr besteht die Notwendigkeit, die Robustheit und Zuverlässigkeit des zu entwickelnden Softwaresystems durch geeignete und in der Praxis bewährte Vorgehensmodelle sicherzustellen. Kernursachen für das Scheitern solcher Softwareprojekte sind:

- Ad-hoc-Anforderungs- und Änderungsmanagement: Erfassen und Zulassen neuer oder geänderter Anforderungen,
- unentdeckte Inkonsistenzen zwischen Anforderungen, Design und Implementierung,
- mangelnde oder ungenaue und undeutliche Projektkommunikation,

- nicht ausreichender Testumfang und
- unnötig komplexe Anwendungsarchitektur.

Diese Faktoren begünstigen das Eintreten folgender Situationen:

- Die Anforderungen werden nur ungenau verstanden.
- Die Flexibilität bei Änderung der Anforderungen während der Projektdurchführung ist ungenügend.
- Einzelne eigenständige Programm-Module arbeiten nicht einwandfrei zusammen.
- Die Software wird schwer zu warten und zu erweitern.
- Ernste Projektfehler werden zu spät erkannt.
- Die Ausführungsgeschwindigkeit ist unzureichend.
- Die zur Fehlerbehebung häufig notwendige Rekonstruktion der Projekthistorie ist nicht vollständig möglich.
- Es treten zu viele Programmfehler auf.
- *Build*- und *Release*-Prozess werden unzuverlässig.

Die in den folgenden Abschnitten vorgestellten Techniken und Ansätze helfen, diese Probleme zu vermeiden. Hierzu zählen der Einsatz der *UML* (Abschnitt 2.4.1), die Befolgung so genannter *Best Practices* (Abschnitt 2.4.2) sowie die Verwendung von Entwurfsmustern (Abschnitt 2.4.3).

2.4.1. *Unified Modelling Language*

Die *Unified Modelling Language (UML)* [35] ist eine Sprache und Notation zur Spezifikation, Konstruktion, Visualisierung und Dokumentation von Modellen für Softwaresysteme und kann als Industriestandard angesehen werden. Beinahe alle Hersteller von Softwaremodellierungswerkzeugen unterstützen die *UML*. Vorgehensmodelle zur Softwareentwicklung basieren teilweise auf der *UML*, denn sie stellt eine definierte Menge von Modellierungskonstrukten mit einheitlicher Notation und Semantik bereit. Einige Diagrammtypen werden nur bei objektorientierter Softwareentwicklung eingesetzt. Neben so genannten Anwendungsfalldiagrammen, Aktivitätsdiagrammen, Kollaborationsdiagrammen, Sequenzdiagrammen und Implementierungsdiagrammen, welche hier nicht näher betrachtet und in dieser Arbeit keine Verwendung finden, kennt die *UML* folgende weitere Diagrammtypen:

- Klassendiagramm: beschreibt zum einen Beziehungen zwischen Klassen, welche durch Vererbung, Realisierung und Assoziation entstehen. Assoziationen beschreiben als Relationen zwischen Klassen die gemeinsame Semantik und Struktur von Objektverbindungen. Zum anderen können in Klassendiagrammen je nach Abstraktionsgrad auch Schnittstellen einzelner Klassen angegeben sein.
- Zustandsdiagramm: zeigt Objektzustände, Zustandsübergänge und Ereignisse.

2.4.2. Softwareentwicklung *Best-Practices*

Im kommerziellen Umfeld haben sich neben dem Befolgen und Umsetzen von Vorgehensmodellen Regeln gefunden, die die Ursachen von Softwareentwicklungsproblemen zuverlässig bekämpfen helfen und sich daher als allgemein anerkannte Strategien durchgesetzt haben. Solche *Best-Practices* sind:

- Es soll ein striktes und genau definiertes Anforderungsmanagement geführt werden.
- Software soll in vielen Iterationen statt nach dem Wasserfallmodell entwickelt werden, siehe Abschnitt 2.4.2.
- Komponentenbasierte Architekturen und Entwurfsmuster sollen eingesetzt werden.
- Software soll visuell entworfen werden.
- Die Softwarequalität soll kontinuierlich überprüft werden.
- Programm-Quelltext-Änderungen sollen genau protokolliert werden.

In der Anforderungs- und Designphase sollen visuelle Modelle in Diagrammform entwickelt werden. Dies erhöht auf der einen Seite die Verständlichkeit, und auf der anderen Seite können unter Zuhilfenahme gängiger visueller Sprachen wie die *UML* auf diese Weise sämtliche zu modellierende Sachverhalte eindeutig kommuniziert werden. Hierunter fallen Anwendungsfall-Diagramme, Klassendiagramme, Zustandsdiagramme und Sequenzdiagramme, welche in Abschnitt 2.4.1 vorgestellt werden.

Es gibt noch einige weitere solcher Regeln, die unbedingt eingehalten werden sollen, auf die hier jedoch nicht näher eingegangen wird.

Iterative Modelle und Wasserfall-Modelle

Ein Wasserfall-Modell, in der Literatur auch häufig Top-Down-Entwicklungsmodell genannt, legt fest, dass die Projektphasen Anforderungsanalyse, Entwurf, Implementierung, Programm-Einheiten-Test, Subsystem-Test und System-Test jeweils

nur einmal durchlaufen werden und dies in genau der soeben genannten Reihenfolge. In solch einem Modell ist nicht vorgesehen, dass sich Anforderungen mit der Zeit ändern können und dass Entwurfs- und Anforderungsfragen während der Implementierung erneut überdacht werden. So wirken sich Fehler in frühen Phasen stark auf nachfolgende Phasen aus, und damit steigt das Projektrisiko mit der Zeit stark.

Iterative Vorgehensmodelle dagegen verfolgen das Ziel, genau jene Schwächen des Wasserfall-Modells zu umgehen mit dem einfachen Ansatz, das vollständige Wasserfall-Modell in vielen Iterationen zu wiederholen. Hierbei werden zum einen die Ergebnisse einer Iteration ausgewertet und diese Erkenntnisse in der nächsten Iteration berücksichtigt. Entscheidend ist jedoch zum anderen, dass jede Iteration einen vollständigen und in sich geschlossenen Entwicklungszyklus darstellt, an dessen Ende immer eine lauffähige Version des zu erstellenden Softwaresystems steht. Es stellt sich hierbei nicht als Nachteil heraus, dass für jede Phase einer Iteration weniger Zeit zur Verfügung steht als bei dem Wasserfallmodell. Beispielsweise kostet die Anforderungsanalyse weniger Zeit, wenn ihr Ziel nicht die Erfassung aller Details der Anforderungen ist. Ein Vertreter dieser iterativen Vorgehensmodelle ist der *Rational Unified Process* [30].

2.4.3. Entwurfsmuster

Muster helfen beim Entwurf beliebiger komplexer Systeme. In der Softwareentwicklung spiegeln Entwurfsmuster teilweise Softwareentwicklungs-Paradigmen wider, namentlich Modularisierung, Objektorientierung und Aspektorientierung.

Die hier vorgestellten Entwurfsmuster, welche umfassend in [14] dargestellt werden, beschreiben einfache und elegante Lösungen für in der vorliegenden Arbeit relevante Probleme des objektorientierten Entwurfs. Ein Entwurfsmuster stellt eine Abstraktion von vielen gleichartigen konkreten Problemlösungen dar, welche im Laufe der Zeit gefunden wurden. Die Anwendung von Entwurfsmustern hat zum Ziel, den Grad an Wiederverwendbarkeit und Flexibilität von Programm-Quelltext zu erhöhen, vor allem aber dem erfahrenen Softwareentwickler das Einarbeiten in fremden Programm-Quelltext zu erleichtern. Letzteres gelingt, weil die bekanntesten Entwurfsmuster allgemein anerkannt sind und häufig angewendet werden; sie beschreiben einen guten Entwurfsstil. Im Folgenden werden diejenigen Entwurfsmuster beschrieben, die das Design und die Implementierung des *KIEL*-Editors grundlegend bestimmen.

Zustands-Entwurfsmuster

Dieses Entwurfsmuster kapselt Objekt-Verhalten in Abhängigkeit von Programmzuständen. Nach außen scheint ein Objekt sein Verhalten, also seine Klasse, zu ändern, wenn sich sein interner Zustand ändert. Dieser Mechanismus wird umgesetzt, indem Zustandsklassen und eine Schnittstelle definiert werden. Die Schnittstelle wird sowohl von dem Objekt, welches sein Verhalten ändern soll – im Folgenden

Kontext genannt – als auch von jeder Zustandsklasse – im Folgenden *Konkreter-Zustand* genannt – implementiert. Die Anwendung kommuniziert nur mit dem Kontext und dieses Objekt leitet alle Anfragen weiter an ein Konkreter-Zustand-Objekt. Ändert sich nun der interne Zustand des Kontextes, so wird das Konkreter-Zustand-Objekt ausgetauscht gegen ein anderes, welches zum aktuellen internen Zustand passt.

In diesem Entwurfsmuster ist das *Delegator*-Entwurfsmuster enthalten, welches bei der Kommunikation zwischen Partnern eine Indirektion vorsieht, in der Anfragen weitergeleitet werden und Aufgaben tatsächlich nicht vom anfrageempfangenden Objekt ausgeführt werden, sondern von dem Objekt, an welches die Anfragen delegiert werden.

Das Zustands-Entwurfsmuster gruppiert Funktionalitäten nicht wie in objektorientierter Programmierung üblich nach Zuständigkeiten sondern nach Zuständen. Dies ist eine geeignete Sichtweise, wenn hiermit auf große mehrteilige Bedingungsanweisungen, die vom Objektzustand abhängen, verzichtet werden kann.

Aus Implementierungssicht ist zu beachten, dass nur der Kontext die Konkreter-Zustand-Klassen referenziert und verwendet. Vorteile dieses Entwurfsmusters sind zum einen, dass leicht neue Zustände modelliert und hinzugefügt werden können, ohne die bestehende Implementierung zu ändern und zum anderen, dass Zustandsübergänge explizit repräsentiert werden. Ein Zustandsübergang ist nicht mehr abhängig von verschiedenen Parametern, sondern er hängt allein von der Referenz einer Variable auf ein Konkreter-Zustand-Objekt ab. Einsatz findet das Zustandsmuster z. B. in der Implementierung von Editoren.

Befehl-Entwurfsmuster

Bei diesem Entwurfsmuster werden Befehle oder Benutzeraktionen als Objekte gekapselt. Diese Objekte können dann in eine Warteschlange gestellt und sequentiell abgearbeitet oder auch protokolliert werden. Solche Objekte eignen sich bei Verwendung von Klassenbibliotheken für Benutzungsschnittstellen, da der Aufrufer solch eines Befehlsobjektes, in der Regel ein Element der Benutzeroberfläche, nichts über die Spezifika des eigentlichen Befehlsaufrufes sondern nur eine standardisierte Schnittstelle zum Befehlsaufruf kennen muss.

Implementiert wird dieses Muster, indem zunächst eine abstrakte Klasse oder eine Schnittstelle (engl. *Interface*) definiert wird, welche genau eine Operation besitzt. Zu jedem Befehl wird dann eine diese Schnittstelle implementierende Klasse entworfen, deren Operation genau den zu kapselnden Befehl ausführt. In Java heißt diese Schnittstelle `javax.swing.Action` und findet Verwendung bei allen Steuerelementen wie Buttons und Menü-Elementen einer Benutzeroberfläche.

Model-View-Controller-Entwurfsmuster

Das *Model-View-Controller*-Entwurfsmuster (*MVC*) verfolgt den Ansatz, eine Anwendung mit graphischer Benutzeroberfläche aufzuteilen in ein Modul, welches

die Reaktion auf Benutzereingaben definiert (engl. *Controller*), in ein weiteres Modul, welches das Datenmodell und Operationen auf diesem beinhaltet (engl. *Model*) und in eine Komponente für die Anzeige der graphischen Benutzeroberfläche (engl. *View*). Sogar eine einzelne Komponente einer Benutzeroberfläche kann hierbei selbst als eine Anwendung mit graphischer Benutzeroberfläche aufgefasst werden. Daher sind z. B. in Java alle Komponenten für Benutzeroberflächen nach dem *MVC*-Muster aufgebaut.

Der entscheidende Punkt bei diesem aus drei Teilen bestehenden System liegt in der definierten Kommunikation zwischen den Teilen. Die Kommunikation verläuft nur unidirektional und nur von *View* nach *Controller*, von *Controller* nach *Model* und von *Model* nach *View*. Daneben ist denkbar, dass auch von außen das *Model* manipuliert werden kann. Der *Controller* ist häufig nach dem Befehl-Entwurfsmuster implementiert, was zur Folge hat, dass der *Controller* austauschbar ist, ohne die *View* oder das *Model* anpassen zu müssen. Auch die *View* lässt sich ohne Änderung von *Controller* und *Model* auswechseln. Der wesentliche Vorteil bei Verwendung dieses Entwurfsmusters liegt wegen der schmalen Kommunikationswege und der Modularisation allerdings in der Verringerung der Komplexität der Anwendung. Des Weiteren ist leicht realisierbar, mehrere *Views* nebeneinander existieren zu lassen, welche auf dem gleichen *Model* basieren und daher ohne Synchronisationsaufwand die gleichen Daten in gleicher oder unterschiedlicher Repräsentation anzeigen.

Die Kommunikation zwischen *View* und *Controller* erfolgt nach dem Befehl-Entwurfsmuster, die Kommunikation zwischen *Model* und *View* hingegen nach dem Beobachter-Entwurfsmuster: Die *View* meldet sich als einer von potentiell mehreren Beobachtern (engl. *Subscriber*) der Änderungen am *Model* (Beobachteter, engl. *Publisher*) an, und das *Model* wird dann bei jeder Zustandsänderung alle Interessenten benachrichtigen. Dieses Beobachter-Entwurfsmuster wird daher gelegentlich auch *Publisher-Subscriber*-Entwurfsmuster genannt. Zentrales Element des Beobachter-Entwurfsmusters ist die Schnittstelle, welche alle Beobachter implementieren, um so dem Beobachteten eine einheitliche Zugriffsmöglichkeit zur Verfügung stellen zu können.

Die Separation in *View* und *Controller* ermöglicht es ferner, die Reaktion eines *Views* auf Benutzungseingaben sogar während der Laufzeit zu ändern. Erreicht wird dies, indem ein Teil des Controllers nach dem Zustands-Entwurfsmuster implementiert werden kann.

3. Stand der Technik

Im vorangegangen Kapitel wurde die Terminologie der *Statecharts* eingeführt und der heutige Stand der Technik bezüglich des methodischen Vorgehens in der Softwa-re-Entwicklung aufgezeigt. In diesem Kapitel hingegen wird der Stand der Technik im Hinblick auf am Markt verfügbare Software-Produkte vorgestellt. Hierbei werden zum einen existierende *Statechart*-Editoren auf ihre Funktionalität hin untersucht, um eine Einordnung des in dieser Arbeit zu entwickelnden Editors zu ermöglichen und eine Vorgabe an wünschenswerten Funktionsmerkmalen zu finden. Zum anderen werden *Graph-Editing-Frameworks* dahingehend analysiert, ob und inwieweit sie für die Realisierung des zu entwickelnden Editors genutzt werden können.

3.1. Analyse existierender *Statechart*-Editoren

Wie in Abschnitt 2.3.2 erwähnt, existieren eine Reihe von *Statechart*-Werkzeugen, welche auf einer graphischen Notation für *Statecharts* basieren und daher auch graphische Editoren zur Bearbeitung von *Statechart*-Diagrammen bereitstellen. Die Vorteilhaftigkeit des graphischen Modellierens von *Statecharts* im Gegensatz zu Ansätzen in Textform beschreibt [18]. Im Fokus der in diesem Kapitel vorgenommenen Analyse von *UML*- und *Statechart*-Werkzeugen wird die Erfassung von interessanten Bearbeitungs- und Gestaltungshilfen für *Statechart*-Diagramme sein, welche die Editoren der vorgestellten Werkzeuge bieten. *Statechart*-Editoren müssen grundlegende Bearbeitungsfunktionen zur Verfügung stellen, welche auch von anderen Diagramm-Editoren her bekannt sind (Abschnitt 3.1.1). Von Interesse für die vorliegende Arbeit ist dabei zum einen, welche Bearbeitungsfunktionen der Benutzer von einem *Statechart*-Editor mindestens erwarten kann, weil jedes Werkzeug sie anbietet. Zum anderen ist aufschlussreich, mit welchen Merkmalen sich die verschiedenen Editoren von ihrer Konkurrenz absetzen und welche Funktionen sie zurzeit noch nicht besitzen. Fehlende Funktionsmerkmale eignen sich im Besonderen zur Umsetzung im *KIEL*-Editor, *KIEL* als Alternative im Umfeld dieser Produkte platzieren zu können.

3.1.1. Grundlegende Funktionen von *Statechart*-Editoren

Sämtliche der hier analysierten *Statechart*-Editoren verfügen über grundlegende Funktionalitäten, welche im Folgenden aufgelistet werden und in Kapitel 4 näher erläutert werden:

- syntaxgerichtetes Editieren,
- Beschriftungsmöglichkeiten für Diagrammelemente und Eingabe von Diagrammelement-Beschriftungen direkt auf der Zeichenfläche ohne Dialogfenster (engl. *Inline-Editing*),
- Löschen, Verschieben und Größenändern von Diagrammelementen,
- rasterorientiertes Bearbeiten,
- relatives Ausrichten von Diagrammelementen (engl. *Alignment*),
- *Cut-and-Paste*,
- Export nach *JPG*, *EPS* und in andere Formate,
- Ausdruck und
- kontextsensitive Mauszeiger zur Anzeige der aktuell möglichen Benutzeraktionen.

3.1.2. *Esterel-Studio*

Esterel [5] ist eine Spezifikationssprache für eingebettete Systeme (Abschnitt 2.1, welche unterstellt, dass eingebettete Systeme sowohl synchron als auch deterministisch sind. *Esterel-Studio* [12] ist dagegen eine auf *Esterel* basierende kommerzielle integrierte Entwicklungsumgebung zur Entwicklung eingebetteter Systeme. Sie bietet die Möglichkeit, *Safe-State-Machine*-Modelle (Abschnitt 2.3.3) zu erstellen. Die formale Spezifikation eines Reaktiven Systems kann dabei graphisch mittels *SSM*-Formalismus oder in Form von *Esterel*-Programm-Quelltext definiert werden. Hierbei ist es möglich, *Esterel*-Programm-Quelltext in graphische Elemente einer *SSM* einzubetten und damit graphische Elemente mit Textelementen zu mischen. Nachdem ein Modell vollständig entworfen wurde, kann dieses Modell auf seine syntaktische Korrektheit hin untersucht werden, indem überprüft wird, ob alle Definitionen des entworfenen Systems konsistent sind. Daneben wird zusätzlich ein *Causality-Problem-Checker* angeboten, welcher das Modell auf Determinismus hin überprüft.

Im Wesentlichen bietet diese Software jedoch die Möglichkeit, interaktiv *Safe-State-Machines* zu simulieren, ausführbaren Programm-Quelltext zu generieren und Berichte zu erzeugen. Bei der Simulation eines entworfenen Systems kann der Benutzer manuell die Zeitschritte (engl. *Ticks*) hochzählen und in jedem Zeitschritt Eingabesignale setzen.

Der Editor und die Simulationsumgebung von *Esterel-Studio* nutzen die *Multiple-Document-Interface*-Technik: Sie besteht aus einem primären Fenster, in welches mehrere Dokument-Fenster eingebettet sind, die nur innerhalb des primären Fensters verschoben werden können. In einem Dokument-Fenster wird ein Teil der

SSM dargestellt, welcher ein *Composite-State* (siehe 2.3.2) sein kann. Dieser *Composite-State* kann als Modul abgespeichert und an mehreren Stellen der *SSM* wiederverwendet (instantiiert) werden. Mit Hilfe dieser Modulstruktur und unterstützt durch die Fenstertechnik kann der Benutzer den Editiervorgang hierarchisch gliedern. Diese Gliederung wird im primären Fenster in einer Baumansicht dargestellt neben Berichten, Nachrichten und Fehlermeldungen, welche ihrerseits in einem Konsolenbereich angezeigt werden.

Editor-Funktionalitäten

Zentrales Element der Benutzeroberfläche zum Erstellen von *Safe-State-Machines* ist die Werkzeugleiste (engl. *Toolbar*) zum Einfügen von Diagrammelementen, siehe Abbildung 3.1, welche folgende Buttons von links nach rechts besitzt: Wechsel in den Selektionsmodus, Einfügen eines *Simple-State*, eines Moduls, eines *OR-States*, eines *OR-States* mit Inhalt in Textform, eines Textblocks, einer Transition, eines *Initial-States*, eines *Suspend*, eines *Deep-History-States*, eines *Choice* und einer horizontalen und vertikalen *Delimiter-Line*. In *OR-States* mit Inhalt in Textform kann *Esterel*-Programm-Quelltext eingefügt werden, welcher den Inhalt dieses *OR-States* beschreibt.



Abbildung 3.1.: Die *Esterel-Studio*-Werkzeugleiste für das Erzeugen von Diagrammelementen

Zur Bearbeitung von *Safe-State-Machines* werden neben den in Abschnitt 3.1.1 genannten grundlegenden Funktionalitäten folgende Operationen angeboten:

- Bearbeitungsschritte können rückgängig gemacht werden (engl. *Undo-and-Redo*).
- Der innere Bereich eines *Composite-States* kann vergrößert werden, ohne dass sich die Grenzen dieses Zustands ändern. Der *Composite-State* erhält dann Scrollleisten (engl. *Scrollbars*). In diesem scrollbaren Bereich kann gezoomt und navigiert werden.
- *Composite-States* können komplett eingeklappt werden, so dass nur noch die Titelleiste zu sehen ist.
- *Composite-States* können per *CTRL*-Taste + Doppelklick in einem eigenen Fenster geöffnet werden.
- Zustände können um Stufen in der Hierarchie nach oben und unten verschoben werden, indem entweder diese Zustände in einen *OR-State* eingeschlossen

werden oder diese aus dem sie umgebenden *Composite-State* herausgenommen werden.

- Für Zustände kann ein Typmorphing (Abschnitt 4.17) durchgeführt werden.
- Für Zustände kann die Eigenschaft *Ist Endzustand* gesetzt werden, jedoch nur, wenn sie keine ausgehenden Transitionen besitzen.
- Deklaration von *On-Entry-Actions*, *On-Inside-Actions* und *On-Exit-Actions*.
- Ein *Initial-State* kann auskommentiert werden, was zur Folge hat, dass der gesamte *OR-State* deaktiviert wird und so bei der Simulation und Programm-Quelltext-Generierung unbetrachtet bleibt.
- Lokale Variablen werden für *Initial-States* deklariert.
- Transitionsendpunkte sind veränderbar.
- Prioritäten können für Transitionen definiert werden, welche abgehen von *Initial-States*, *Conditional-States* und *Simple-States*. Diese Prioritäten sind in einem Prioritäteneditor änderbar. Dort werden die Prioritäten in einer Liste verwaltet.
- Jede Transition ist standardmäßig eine *Strong-Abortion*. Erst nach dem Erzeugen kann man den Transitionstyp ändern (Abschnitt 4.17).
- Verschieben mehrerer Elemente ist möglich mit der *CTRL*-Taste + Mausklick oder indem ein diese Elemente umgebendes Rechteck gezeichnet wird, welches dann mitsamt seinem Inhalt verschoben werden kann.
- Schriftart und Farbe sind einstellbar für Diagrammelemente und ihre Beschriftungen.
- Kommentare sind überall platzierbar.
- Ein globales *Suchen und Ersetzen* ist möglich.
- Editieren von Transitionsbeschriftungen ist über ein Dialogfenster möglich, welches Eingaben für *Trigger*, *Condition* und *Action* separiert.

3.1.3. *Matlab / Simulink / Stateflow*

Matlab ist ein weit verbreitetes Programm im Bereich des wissenschaftlichen numerischen Rechnens. Es handelt sich hierbei um ein interaktives System mit Matrizen als Basis-Datenelement und bietet graphische Darstellungsmöglichkeiten, welche auch durch benutzerdefinierte Benutzeroberflächen erweitert werden können. Es können Befehle ähnlich wie bei einem Taschenrechner direkt über eine Kommandozeile eingegeben werden oder aber auch in Form von *Scripts*, geschrieben in der *Matlab*-eigenen Programmiersprache.

Bei *Simulink* [33] handelt es sich um eine Funktionsbibliothek für *Matlab* zur Simulation regelungstechnischer Problemstellungen und besitzt eine auf *Matlab* aufgesetzte Benutzeroberfläche, in deren graphischem Editor (Differential-) Gleichungssysteme als hierarchische Signalflusspläne modelliert werden können. Aus diesen Modellen kann dann *C*-Programm-Quelltext generiert werden, um diesen z. B. für Echtzeitsysteme zu kompilieren. Ähnlich wie andere Werkzeuge, welche Programm-Quelltext generieren, gehört *Simulink* damit auch zur Gruppe der *Rapid-Prototyping-Tools*. Daneben können die angefertigten Modelle auch simuliert werden. Sowohl *Matlab* als auch *Simulink* sind Produkte der Firma *The Math Works* [47].

Stateflow wiederum stellt eine graphische Erweiterung zu *Simulink* dar und ermöglicht die Modellierung und Syntaxprüfung von *Statecharts* und durch Simulink auch die Simulation dieser Modelle. Während der Simulation können dabei alle verfügbaren Matlab-Funktionen benutzt werden, und es steht für die Simulation ein *Debugger* zur Verfügung, in welchem *Breakpoints* an beliebigen Diagrammelementen und Ereignissen gesetzt werden können. Voraussetzung für die Benutzung von *Stateflow* ist die *Matlab / Simulink*-Umgebung.

Jedes *Statechart*-Werkzeug hat seine eigene Darstellungsform für *Statechart*-Diagrammelemente. Abbildung 3.2 zeigt beispielhaft ein *Statechart* mit seinen Diagrammelementen in *Stateflow*-Notation. Abbildung 3.3 zeigt die Art der Darstellung von *State-Actions*.

Editor-Funktionalitäten

Die Werkzeugleiste ist Ausgangspunkt zum Einfügen von Diagrammelementen, siehe Abbildung 3.4, welche folgende Buttons von oben nach unten besitzt: Hinzufügen eines *States*, eines *History-States*, eines *Initial-States* und eines *Conditional-Pseudo-States*. In *Stateflow* wird bei Hinzufügen eines *Initial-States* auch gleich eine *Initial-Arc* angehängt, welche mit einem Zustand verbunden wird.

Zur Bearbeitung von *Statecharts* werden neben den in Abschnitt 3.1.1 genannten grundlegenden Funktionalitäten folgende Operationen angeboten [34]:

- Bearbeitungsschritte können rückgängig gemacht werden.
- Hinzufügen von Diagrammelementen ist möglich im *Single-Creation*-Modus und im *Multiple-Creation*-Modus: Durch Doppelklick auf ein Diagrammelement in der Werkzeugleiste lassen sich mehrere Instanzen dieses Elementtyps hinzufügen, ohne wiederholt diesen Typ in der Werkzeugleiste anzuwählen.
- Allen Diagrammelementen können Beschreibungen hinzugefügt werden.
- Beschriftungsschriftarten und Farben können für alle Diagrammelemente geändert werden.
- Der Zoomfaktor für *Zoom-In* und *Zoom-Out* ist änderbar.

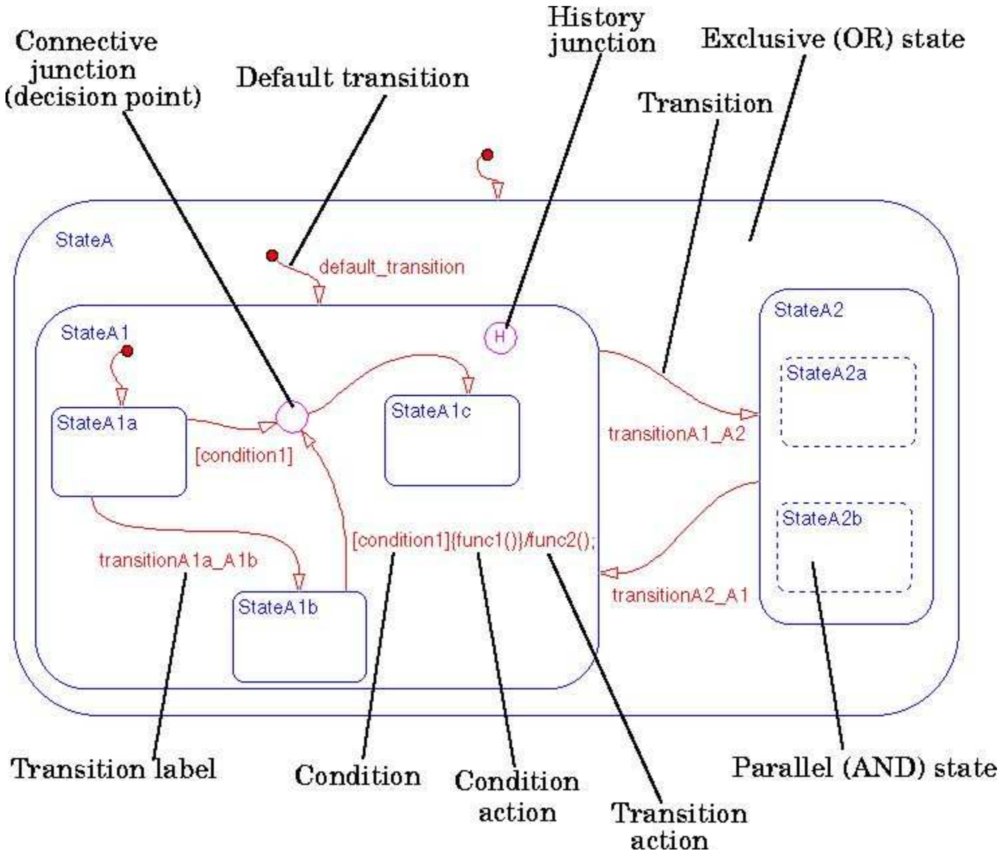


Abbildung 3.2.: Ein Beispiel-*Statechart* aus *Stateflow*

- Beim Anlegen von Zuständen wird nicht zwischen *Simple-States* und *Composite-States* unterschieden. Sobald in einem *Simple-State* ein weiterer Zustand platziert wird, findet ein automatisches Typmorphing nach *OR-State* statt (Abschnitt 4.17).

- Über ein Kontext-Menü legt der Benutzer fest, ob es sich bei einem *Composite-State* um einen *AND-State* oder einen *OR-State* handeln soll.

- Parallele *Substates* eines *AND-States* beinhalten zusätzlich in der rechten oberen Ecke eine Nummer, welche die Aktivierungsreihenfolge beim Betreten des *AND-States* angibt.

- Gruppieren eines Zustands: Ist ein Zustand als *gruppiert* markiert, so wird er mitsamt seinem Inhalt als graphische Einheit behandelt, z. B. wird ein *gruppierter* Zustand mitsamt seinen Unterzuständen verschoben; nicht so ist dies hingegen bei *ungruppierten* Zuständen.

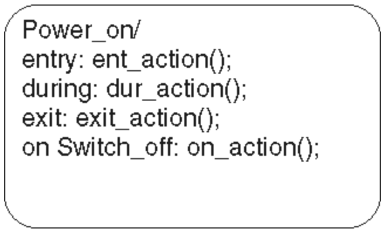


Abbildung 3.3.: Anzeige der Actions eines Zustands in *Stateflow*

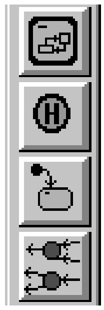


Abbildung 3.4.: Die *Stateflow*-Werkzeugleiste für die Auswahl von Diagrammelementen

- Der Inhalt eines Zustands kann versteckt und auch wieder sichtbar gemacht werden, indem die entsprechende Zustandseigenschaft der Sichtbarkeit im Kontext-Menü eingestellt wird.

- *Composite-States* können als Teildiagramme deklariert werden, welche als eigenständige Diagramme bearbeitet werden können. Dies wird in *Stateflow* auch *Boxing* genannt.

- Transitionsbeschriftungen können verschoben werden.

- Die Größe von Transitions-Pfeilspitzen ist änderbar.

- *Statecharts* werden in Bibliotheken organisiert. Eine Änderung in einem Bibliotheks-*Statechart* wirkt sich auf alle *Statecharts* aus, welche das geänderte *Statechart* nutzen.

- Zur Verfügung steht ein so genannter *Stateflow-Explorer*, welcher ein Verzeichnis darstellt, in dem alle Diagrammelemente mitsamt allen Ereignissen in einer Baumstruktur angezeigt werden. Die Änderung der Sichtbarkeit von Ereignissen muss über diesen *Explorer* erfolgen, Hierarchie-Änderungen und das Anlegen und Löschen von Elementen dagegen *können* über diesen *Explorer* durchgeführt werden.

3.1.4. *ArgoUML*

ArgoUML [3] baut auf dem *GEF*-Rahmenwerk (Abschnitt 3.2.3) auf und gehört zu der Familie der *UML*-Modellierungswerkzeuge, welche eine Teilmenge der in Abschnitt 2.4.1 genannten Diagrammformen zur Erstellung anbieten. Es handelt sich bei *ArgoUML* um ein *Open-Source*-Projekt und ist daher kostenfrei nutzbar. Dafür wird weder Support angeboten noch ist Dokumentation verfügbar, welche allgemeinen Standards bezüglich Vollständigkeit und Nutzbarkeit genügt. Auch wird kein Hilfesystem angeboten, welches interaktiv aus der Benutzeroberfläche aufgerufen werden kann [26]. Im Gegensatz zu anderen *Statechart*-Werkzeugen bietet *ArgoUML* ausschließlich Möglichkeiten zum graphischen Editieren von Diagrammen, eine Syntaxprüfung und Programm-Quelltext-Erzeugung nach C++, PHP und Java zum Zwecke des *Forward-Engineering*. Darüber hinaus kann das Modell auf Einhaltung der über die *Object-Constraint-Language* vom Benutzer definierten Beschränkungen (engl. *Constraints*) hin geprüft werden.

ArgoUML beinhaltet so genannte *Design-Critics* [42]. *Design-Critics* werden in [3] beschrieben als einfache Agenten, welche im Hintergrund ständig das Design des Benutzers analysieren, um gegebenenfalls Verbesserungsvorschläge unterbreiten zu können. Diese Vorschläge umfassen Hinweise auf syntaktische Fehler, noch nicht fertig gestellte Entwurfsbereiche und Verstöße gegen Programmierstil-Richtlinien sowie *Best-Practices* der Softwareentwicklung (Abschnitt 2.4.2). Einige dieser *Design-Critics* bieten die automatische Durchführung einer Entwurfsverbesserung an. Sämtliche Verbesserungsvorschläge und Hinweise werden lediglich in einer *To-Do*-Liste aufgeführt und unterbrechen den Benutzer während des Bearbeitens nicht. Diese *To-Do*-Liste kann vom Benutzer frei ergänzt werden. Sowohl *Design-Critics* als auch *To-Do*-Liste stellen neben dem Angebot an verschiedenen Ansichten auf das Modell und Check-Listen eine Anwendung von Erkenntnissen aus der kognitiven Psychologie dar.

Editor-Funktionalitäten

Ergänzend zu den in Abschnitt 3.1.1 genannten grundlegenden Funktionalitäten werden in *ArgoUML* zur Bearbeitung von *Statecharts* folgende Operationen angeboten:

- Diagramme können exportiert werden nach *PS*, *EPS*, *PGML*, *SVG* und *XMI*.
- Optional ist ein Raster anzeigbar und rasterorientiertes Bearbeiten ist möglich.
- Alle Diagrammelemente werden zusätzlich in einer Baumansicht angezeigt.
- Die Baumansicht ist wahlweise diagramm-, transitions- oder zustandszentriert.
- Ein *Undo-and-Redo* von Navigations- und Selektionsvorgängen ist möglich.

- Ausrichtung von Diagrammelementen relativ zueinander mittels eines Balkens, welcher wie ein Besen alle erfassten Diagrammelemente an seiner Kante verschiebt (engl. *Broom-Tool*).
- Zur Verfügung stehen neben den *Statechart*-Diagrammelementen beliebige Ovale und Polygone, welche zu Dokumentationszwecken genutzt werden können.
- Am Rahmen eines selektierten Zustands sind zwei Buttons sichtbar, welche der Erzeugung von eingehenden und ausgehenden Transitionen dienen (engl. *Rapid-Buttons*).
- Zustände können auch mit den *Cursor*-Tasten verschoben werden.
- Im Selektionsmodus kann ein Rechteck gezeichnet werden, dessen Inhalte nach Beendigung des Zeichnens selektiert sind (engl. *Marquee-Selection*).
- Zusätzlich zu den üblichen Alignment-Funktionen wird angeboten das Verteilen von horizontalem und vertikalem Platz zwischen Diagrammelementen.

3.1.5. *Poseidon For UML*

Poseidon For UML von der Firma *Gentleware* ist eine auf *ArgoUML* basierende kommerzielle Software und gehört damit ebenfalls zur Familie der *UML*-Modellierungswerkzeuge. Es besitzt alle Funktionsmerkmale von *ArgoUML*, bietet jedoch einige weitere Bearbeitungshilfen, wie z. B. den Import von *Rational Rose*-Modellen, *Roundtrip-Engineering* für Java und Layoutmechanismen. Diese Layoutmechanismen können als *Plugins* von verschiedenen Herstellern integriert werden. Angeboten wird derzeit z. B. ein *Plugin* von der Firma *yWorks* [52].

Editor-Funktionalitäten

Ergänzend zu den in Abschnitt 3.1.1 genannten grundlegenden Funktionalitäten und denen von *ArgoUML* (Abschnitt 3.1.4) werden in *Poseidon For UML* zur Bearbeitung von *Statecharts* folgende Operationen angeboten:

- Bearbeitungsschritte können rückgängig gemacht werden.
- Diagrammausschnitte können als Graphik in andere Anwendungen kopiert werden.
- *Reverse-Engineered* Diagramme und graphisch modellierte Diagramme können gelayoutet werden. Wird nach einem generierten Layout das Diagramm verändert, so kann hiernach statt einer neuen Layoutgenerierung ein aktualisierendes Layout durchgeführt werden, welches ein neues Layout unter Beachtung des letzten generierten Layouts berechnet. Ein auf diese Weise erzeugtes Layout ähnelt dem vorherigen Layout größtmöglich auf Kosten sich eventuell kreuzender Transitionen.

- Eine Übersichtsansicht des gesamten Diagramms und und eine Eigenschaftsansicht des aktuell selektierten Elements wird angeboten in einem separaten in die Benutzeroberfläche eingebundenen Bereich.
- Die Wahl des bearbeitbaren sichtbaren Bereichs des Diagramms und dessen Zoomfaktor ist über die Übersichtsansicht steuerbar.
- Die Sichtbarkeit von Benutzeroberflächen-Elementen ist konfigurierbar.
- Schriftarten und Farben können individuell für jedes Diagrammelement eingestellt werden.

3.1.6. Ergebnis

Folgende Bearbeitungs- und Gestaltungshilfen werden von keinem der hier vorgestellten *Statechart*-Werkzeuge angeboten:

- automatisches Layout während des Platzierens von Diagrammelementen (Abschnitt 4.13),
- gleitende Strukturveränderung bei Layoutmaßnahmen (Abschnitt 4.7),
- Abspielen von Bearbeitungsschritten (Abschnitt 4.9),
- mehrschichtige Übersichtsdarstellungen (Abschnitt 4.10) und
- Bearbeitungsschritt-Historie (Abschnitt 4.23).

Diese Funktionalitäten werden in dieser Arbeit als nützlich eingestuft und bieten sich daher zur Implementierung im *KIEL*-Editor an. Andere Funktionsmerkmale wie z. B. die *Design-Critics* des Produktes *ArgoUML* werden ebenfalls als sinnvoll erachtet, liegen allerdings nicht im Fokus von *KIEL*, siehe hierzu Abschnitt 1.1.

3.2. Evaluierung verfügbarer *Graph-Editing-Frameworks*

Ein Rahmenwerk (engl. *Framework*) ist ein Softwaresystem, welches nur partiell vollständig und damit nicht allein lauffähig ist, welches aber die Architektur für eine spezifische Klasse von ähnlichen Softwaresystemen definiert und auf diese Weise zusätzlich problemspezifische Funktionalitäten bietet.

Ein Ziel der vorliegenden Arbeit ist es, die am Markt verfügbaren Rahmenwerke für Graph-Editoren dahingehend zu untersuchen, inwieweit diese zur Implementierung eines *Statechart*-Editors genutzt werden können. Durch Verwendung solcher Rahmenwerke kann der Entwicklungsaufwand zur Realisierung häufig wiederkehrender grundlegender Funktionen von Graph-Bearbeitungs-Werkzeugen deutlich verringert werden. Einige Rahmenwerke werden überblickartig und ausführlich in [24] dargestellt.

Als *Graph-Editing-Frameworks* werden solche Softwareprodukte bezeichnet, welche Benutzerinteraktion, Graphdarstellung und Graph-Bearbeitungsmechanismen bereitstellen. Im *Java*-Umfeld sind solche Rahmenwerke häufig *Swing*-orientiert und setzen das *MVC*-Muster (Abschnitt 2.4.3) um.

3.2.1. Anforderungen

Jedes der in Abschnitt 3.2.2 untersuchten Rahmenwerke bietet folgende grundlegende Graph-Bearbeitungsfunktionen:

- Darstellung von Knoten mit unterschiedlichen Formen,
- Verbinden zweier Knoten mit einem gerichteten Pfeil,
- Pfeile haften an Knoten beim Verschieben von Knoten,
- Skalierung der Darstellung,
- *Cut-and-Paste*-Funktionalität und
- Veränderbarkeit der Größen von Knoten.

Alle verfügbaren hier betrachteten Rahmenwerke unterscheiden sich voneinander allerdings bezüglich folgender Eigenschaften, deren Vorhandensein für den *KIEL-Statechart*-Editor notwendig ist:

Graph-Hierarchien: Die Fähigkeit, Hierarchien darzustellen, zeigt sich einerseits in einer baumartigen Modell-Struktur, und andererseits soll ein Verschieben vollständiger Hierarchiezweige möglich sein.

KIEL-Konformität der Darstellung: Das Rahmenwerk soll *Statecharts* genau so anzeigen können wie der *KIEL*-Browser [50].

Live-Preview: Als bedienungsfreundlich zeigt sich die Eigenschaft, ein Element während seines Verschiebens vollständig darzustellen. Der Benutzer sieht auf diese Weise zu jedem Zeitpunkt, wie das Ergebnis des Verschiebens aussieht. Des Weiteren sollen während des Verschiebens eines Diagrammelements andere Elemente änderbar sein, um die in Abschnitt 4.13 beschriebenen Layouthilfen implementieren zu können.

Layers: Die Reihenfolge übereinander liegender Diagrammelemente soll variabel sein, sofern die Elemente nicht als transparente Objekte dargestellt werden. Anderenfalls würde ein unter einem größerem Diagrammelement liegendes kleineres Element verdeckt werden.

Model-View-Controller-Konformität: Verschiedene Sichten auf das *Statechart* sollen möglich sein, ohne Aktualisierungsmechanismen implementieren zu müssen. Im *KIEL*-Editor sollen bei einer Datenmodell-Änderung – z. B. veranlasst durch eine Benutzeraktion – die Bearbeitungssicht und die Übersichtsdarstellung automatisch aktualisiert werden, siehe Abschnitt 2.4.3, so dass beide Sichten zu jedem Zeitpunkt eine Repräsentation des aktuellen *Model*-Zustands zeigen.

Verschiedene Pfeil-Stile: Es sollen Spline-Kurven und Polygone gezeichnet werden können.

Undo-and-Redo-Funktionalität: Bearbeitungsschritte sollen rückgängig gemacht werden können.

Layout: Vorteilhaft sind in das Rahmenwerk integrierte Layout-Algorithmen, welche die Implementierung der in Abschnitt 4.13 beschriebenen Mechanismen unterstützen können.

Selektive Sichtbarkeit von Diagrammelementen: Das Kollabieren und Expandieren von Knoten erfordert eine variable Sichtbarkeit von Diagrammelementen, siehe Abschnitt 4.5.

Beschriften von Knoten und Pfeilen soll möglich sein.

Syntaxgerichtetes Editieren: Pfeile müssen immer Anfangs- und End-Knoten haben.

Um jedoch eines der getesteten Rahmenwerke erfolgreich einsetzen zu können, sind darüber hinaus auch die folgenden projekttechnischen Eigenschaften von Bedeutung [17].

Projektaktualität: Die schnell fortschreitende Entwicklung im Bereich der *Java*-Programmierung fordert ständige Aktualität von Rahmenwerken, um kompatibel zur aktuellen *Java*-Version zu sein.

Projektaktivität: Es soll eine rege Entwicklungstätigkeit erkennbar sein, z. B. hoch frequentierte Benutzer-Foren oder eine hohe Fehlerbearbeitungsrate, welche ihrerseits ersichtlich ist in entsprechenden Fehler-Datenbanken (engl. *Bug-Tracking-Tools*). Aus einer hohen Aktivität kann auf stabile Versionen und Produktunterstützung geschlossen werden.

Projektkommunikation: Kommunikation mit den Entwicklern oder Benutzern des Rahmenwerks über Foren kann hilfreich bei der Anwendung des Rahmenwerks sein.

Dokumentation: Hierzu sollen Design-Papiere gehören, Unterrichtsmaterialien (engl. *Tutorials* und *How-To's*), Quelltextkommentare (*Javadoc*, engl. *Inline-Code-Comments*) und die Verfügbarkeit des kompletten Quelltextes.

Programm-Quelltext-Umfang: Rahmenwerke werben mit einfachem, schlankem und leicht verständlichem Design, welches sich hervorragend erweitern und an viele Bedürfnisse anpassen lässt. Zur objektiven Bewertung dieser Kriterien eignet sich die Größe des Programm-Quelltextes in Relation zur Funktionalität der Beispiele. Hierbei muss jedoch beachtet werden, dass in manchen Rahmenwerken zusätzlich weitere Standard-Rahmenwerke wie z. B. solche zur *XML*-Bearbeitung enthalten sind.

Verwendete Entwurfsmuster: Entwurfsmuster (Abschnitt 2.4.3) erhöhen generell die Wartbarkeit und Robustheit der Software und senken die Einarbeitungszeit.

Beispiele: Diese dienen der Funktionsanalyse des Rahmenwerks und ergänzen die Dokumentation.

Grad an Flexibilität und Erweiterbarkeit: Dieser lässt sich an der Anzahl und Art der Aufsetzpunkte (engl. *Hooks*) messen. Funktionelle Erweiterungen sollen allein durch Subklassenbildung und/oder das Setzen von Eigenschaften implementiert werden können. Der Programm-Quelltext des Rahmenwerks soll hierfür nicht geändert werden müssen.

Projektstruktur: Die Gliederung in *Add-Ons* und Rahmenwerk-Erweiterungen dient der Übersichtlichkeit und Verringerung der Komplexität. Vollständig auf dem jeweiligen Rahmenwerk basierende *Open-Source*-Produkte, hier z. B. komplette Editoren, können zur Verringerung des Entwicklungsaufwandes führen und zeigen darüber hinaus die Anwendung des Rahmenwerks.

Nichtkommerzielle *Open-Source Java*-API: Der zu entwickelnde Editor soll Bestandteil von *KIEL* werden, welches seinerseits in *Java* geschrieben ist. Zwar ist keine enge Verzahnung des Editors mit dem Rest des Projektes vorgesehen, zumindest soll jedoch der Editor auf der *KIEL*-Datenstruktur operieren können. Daher ist es zweckmäßig, den Editor in *Java* zu entwickeln. Der Editor wie auch das gesamte Projekt wird zu Forschungszwecken entwickelt und ein Ende der Entwicklungstätigkeit ist nicht abzusehen. Darüber hinaus soll das fertiggestellte *Statechart*-Werkzeug auch ohne lizenzrechtliche Restriktionen frei eingesetzt werden. Aus diesen Gründen soll die Wahl auf ein nichtkommerzielles *Open-Source*-Graph-Rahmenwerk fallen. Ferner ist die Änderung des Quelltextes in der Regel bei kommerziellen Produkten nicht möglich. Es gibt über zehn nichtkommerzielle in *Java* geschriebene *Open-Source*-Graph-Rahmenwerke, siehe Abschnitt 3.2.2, daher stellen diese geforderten Eigenschaften keine zu starke Einschränkung in der Auswahl eines Rahmenwerks dar.

Wie sich im nächsten Abschnitt zeigt, stellen die in diesem Abschnitt geforderten Eigenschaften an *Graph-Editing-Frameworks* keine unerfüllbaren Bedingungen bei der Auswahl eines Rahmenwerks dar.

3.2.2. Übersicht über *Graph-Editing-Frameworks*

Insgesamt wurden in dieser Arbeit 16 *Graph-Editing-Frameworks* untersucht, welche als Grundvoraussetzung auf *Java* basieren. Aus diesen Rahmenwerken wurden entsprechend den Anforderungen aus dem vorherigen Abschnitt die vier passendsten Produkte ausgewählt und in diesem Abschnitt tabellarisch einander gegenübergestellt. Diese vier Rahmenwerke sind quelloffen, nicht-kommerziell und sind in einer aktuellen Version verfügbar. Der hier vorgenommene Vergleich dieser Rahmenwerke basiert zum einen auf der Auswertung der zur Verfügung stehenden jeweiligen Dokumentation und zum anderen auf Quelltextanalyse und kleineren Implementierungstests. Die übrigen zwölf Rahmenwerke werden im Abschnitt 3.2.3 kurz vorgestellt.

Das Vorhandensein der in Abschnitt 3.2.1 genannten Merkmale in den hier ausgewählten Rahmenwerken wird in tabellarischer Form im Folgenden ausgewiesen. In der Tabelle 3.1 werden in einer Übersicht funktionale und projekttechnische Merkmale dieser Rahmenwerke miteinander verglichen. Nähere Erläuterungen zu den Angaben in der Tabelle liefert Abschnitt 3.2.3.

Die Symbole in der Tabelle 3.1 bedeuten:

- $\surd$: Ist vorhanden
- $++$: Sehr gut, sehr viel
- $+$: gut, mehrere
- *o*: Aufwand oder Umfang kann nicht genau geschätzt werden
- -: unterdurchschnittlich, gering
- - -: ungenügend, fehlend

Im nächsten Abschnitt werden Erläuterungen zum Inhalt der Tabelle 3.1 gegeben.

3.2.3. Ergebnis

Nachfolgend werden die Vorteile und Nachteile der Rahmenwerke *GEF*, *GVF*, *JGraph* und *GINY* auf Basis der Tabelle 3.1 dargestellt. Das Rahmenwerk *JGraph* überzeugt am meisten sowohl im Funktionsumfang als auch in den projekttechnischen Eigenschaften und eignet sich daher am besten als Grundlage für die Implementierung des *KIEL*-Editors.

GEF - *Graph Editing Framework*

Positiv fällt in diesem Rahmenwerk [13] auf, dass es bekannte Entwurfsmuster wie das *MVC*-Muster (Abschnitt 2.4.3), Befehlsmuster (Abschnitt 2.4.3) und Zustandsmuster (Abschnitt 2.4.3) verwendet, und die Darstellungsebene der Architektur auf den *Java-Swing*-Klassen `javax.swing.JTree` und `javax.swing.JTable`

	GEF	*GVF*	*JGraph*	*GINY*
Funktionale Anforderungen				
Graphhierarchien	- - (jedoch *ArgoUML*)	√	√	√
Darstellung *KIEL*-konform	- -	- -	√	o
Live Preview	√	o	√	- -
Layers	√ (gut)	o	√	√
Model-View-Controller	√	- -	√	- -
Verschiedenartige Pfeil-Stile (Spline, Bezier, Orthogonal)	√	o	√	o
Undo-and-Redo	unvoll- ständig	o	√	- -
Layout	- -	√	√	√
Selektive Sichtbarkeit von Elementen	o	√	√	√
Anhängen von Text	√	o	√	- -
Syntaxgerichtetes Editieren (Pfeile)	√	o	√	o
Projekttechnische Anforderungen				
Projektaktualität	++	-	++	+
Projektaktivität	o	- -	++	- -
Projektkommunikation (Entwickler / Foren / Aktualität)	- o/- -/o	- - - -/-/-	++/++/++	- -/- -/- -
Architektur-Dokumentation (Anz. Seiten / Wertung)	0/- -	- - 16/+	+ 52/++	22+15/+
Tutorial-Dokumentation (Anz. Seiten / Wertung)	0/- -	0/- -	+ 30/+	15/+
Umfang (Größe *Bytecode* in kB)	523	821	135	1680
Verwendete Entwurfsmuster	++	+	+	+
Beispiele (Anzahl/Größe in kB)	4 / 95	0/0	7/54	0/0
Erweiterbarkeit	√	o	√	o
Erweiterungen / Produkte	- -/++	- -/+	++/+	- -/- -
Nichtkommerziell und *Open-Source*	√	√	√	√

Tabelle 3.1.: Funktionale und projekttechnische Anforderungen

basiert. Dem gegenüber werden einige wichtige Funktionalitäten wie z. B. Graph-Hierarchien und *Undo-and-Redo*-Mechanismen nicht bereitgestellt [42], welche allerdings zumindest im auf dem *GEF* aufbauenden Softwareprodukt *ArgoUML* (Abschnitt 3.1.4) implementiert werden. Ebenso unvorteilhaft erscheint die Tatsache, dass in *GEF* Funktionalitäten enthalten sind, welche in ein Rahmenwerk nicht hineingehören, weil sie zu anwendungsspezifisch sind, wie z. B. die Möglichkeit, Makros aufzeichnen und abspielen zu können. Dies führt zu Unübersichtlichkeit und erhöhtem Einarbeitungsaufwand, da die Komplexität des Rahmenwerks mit zunehmender Größe zunimmt.

Aus den Beispielen und der Dokumentation ist nicht ersichtlich, dass verschiedene Darstellungsmöglichkeiten für Diagrammelemente bestehen. Die Dokumentation und Funktionsmerkmal-Liste ist über sechs Jahre alt, neue *Releases* erscheinen hingegen alle 2-4 Monate. Zwar werden E-Mail-Adressen der Entwickler veröffentlicht, jedoch gibt es keine Foren für dieses Projekt und auch auf ein Fehlerberichtswerkzeug (engl. *Bug-Tracking-Tool*) ist nicht zugreifbar, daher kann die Projektaktivität nicht beurteilt werden.

GVF - Graph Visualization Framework

Dieses Rahmenwerk, in [31] und [32] näher beschrieben, besitzt neben 16 Seiten Dokumentation einen vollständig implementierten Graph-Editor, der jedoch undokumentiert ist. Die letzte Version ist sechs Monate alt und die Vorgängerversion wurde im Jahr 2002 freigegeben. Es existiert keine Projektkommunikation und auch Foren sind nicht vorhanden. Zumindest Entwurfsmuster wie *MVC*-Muster (Abschnitt 2.4.3) und Dekorierer-Muster werden eingesetzt. Positiv fällt auch auf, dass die Architektur bezüglich Graph-Struktur und Graph-Darstellung derjenigen von *KIEL* ähnlich ist und dass weitergehende Funktionalitäten wie Layout und Fokus-und-Kontext-Zoom [39] implementiert sind.

JGraph

Das Rahmenwerk *JGraph* [2] zeichnet sich durch eine relativ große Benutzergemeinde aus, welche in mehreren Foren organisiert ist, sowie durch eine hohe Projektaktualität und Aktivität. Neue Versionen des Rahmenwerks werden monatlich freigegeben. Kommunikation mit den Entwicklern ist ebenso möglich wie das Berichten von Fehlern in *Bug-Tracking-Tools*. Das Rahmenwerk ist nicht kommerziell. Allerdings wird durch den Erwerb einer Lizenz eine bevorzugte Unterstützung gewährt.

Die Dokumentation eignet sich zur Erstellung von Applikationen, die auf dieses Rahmenwerk aufsetzen, und auch beispielhafter ständig aktualisierter Programm-Quelltext ist in ausreichender Menge vorhanden. Die Schnittstellen-Definition der Komponenten allerdings ändert sich von Version zu Version, was eine Portierung einer auf *JGraph* basierenden Anwendung auf eine neue *JGraph*-Version erschwert. Zwar erfüllt *JGraph* alle funktionalen Anforderungen, jedoch ist das Rahmenwerk

nicht *Multithreading*-fähig, was im Bereich der gleitenden Strukturveränderung (Abschnitt 4.7) sich als ungünstig erweist.

GINY - Graph Interface library

Bei *GINY*, beschrieben in [8] und [4], handelt es sich um ein Rahmenwerk, welches aus Schnittstellendefinitionen besteht, die durch Bibliotheken anderer Projekte implementiert werden. Eine dieser Bibliotheken ist das *CERN Colt*-Projekt, entwickelt am CERN (Europäische Organisation für Nuklearforschung), welches Datenstrukturen und Operationen für numerische Daten und komplexe Graph-Operationen bereitstellt. Die graphische Seite von *GINY* wird implementiert von *Piccolo*, entwickelt an der Universität Maryland, und enthält eine zoombare Benutzeroberfläche und unterschiedlichste Graph-Knotenarten. Vervollständigt wird *GINY* durch die Bibliothek des *Open-Source*-Projektes *Java Universal Network/-Graph Framework* (*JUNG*), welches Layoutmechanismen auf Graphen zur Verfügung stellt.

Durch diese Projektstruktur ist *GINY* umfangreich und unübersichtlich. Daneben wird Benutzerinteraktion in nur geringem Umfang unterstützt. Insgesamt reichen die Funktionalitäten für eine auf das Rahmenwerk aufbauende Entwicklung eines *Statechart*-Editors nicht aus.

Übersicht über nicht geeignete Rahmenwerke

Nachfolgend sind weitere *Graph-Frameworks* und *Graph-Editing-Frameworks* aufgelistet, welche entweder aus Gründen mangelnder Projektaktualität, aufgrund zu geringen Funktions- oder Dokumentationsumfanges oder aufgrund ihres kommerziellen Charakters grundsätzlich nicht in Frage kommen:

- *VGJ*: Ist zu alt und wird nicht mehr gepflegt [48]
- *GFC*: Es ist keine aktuelle Version verfügbar [6]
- *Grace*: Hierbei handelt es sich um einen *Statechart*-Editor-Generator. Es ist keine ausreichende Dokumentation vorhanden [27].
- *JDigraph*: Es existiert noch keine freigegebene Version (engl. *Release*) sondern nur ein *Alpha-Release*. Außer *Javadoc*-Kommentaren steht noch keine Dokumentation zur Verfügung [49].
- *ToughGraph*: Dieses Rahmenwerk bietet nur die Darstellung von Graphen. Bearbeitungsmöglichkeiten sind nicht vorgesehen [41].
- *JGraphEd*: Unterstützt das Skalieren von Graphteilen, jedoch keine Hierarchien. Die Dokumentation ist nicht ausreichend [20].
- *yFiles*: yFiles erfüllt alle Anforderungen aus Abschnitt 3.2.1. Da es sich allerdings um eine kommerzielle Software handelt, ist eine Verwendung in *KIEL* ungünstig [52].

- *Tom Sawyer Software*: Ebenso wie bei yFiles handelt es sich bei *Tom Sawyer* um ein kommerzielles Rahmenwerk und soll daher in *KIEL* ebenfalls nicht eingesetzt werden [40].
- *Hypergraph*: Für dieses Rahmenwerk ist keine Dokumentation verfügbar [25].
- *Jung*: Gegen eine Verwendung dieses Rahmenwerks sprechen die fehlende Lauffähigkeit der Beispiele, der große Umfang und die fehlende Unterstützung von Graph-Hierarchien. Jung bietet lediglich grundlegende Visualisierungsmöglichkeiten, jedoch keine Editorfunktionalität [38].
- *WilmaScope*: *WilmaScope* unterstützt keine Hierarchien, besitzt eine Benutzeroberfläche zum Editieren und die Möglichkeit einer 3D-Darstellung von Graphen. Eine *KIEL*-konforme Darstellung von *Statecharts* ist allerdings nicht möglich [11].
- *Eclipse GEF*: Dieses Rahmenwerk ist nicht portabel, da es das plattformabhängige *SWT*-Rahmenwerk verwendet. Ausreichend Dokumentation ist vorhanden, allerdings sind die Beispiele nicht lauffähig [23].

4. Anforderungsspezifikation des *KIEL*-Editors

Mit dem zu entwickelnden Editor soll es möglich sein, z. B. das in Abbildung 4.1 gezeigte Referenzbeispiel zu erstellen. Zu den hierfür notwendigen grundlegenden Bearbeitungsfunktionen sollen darüber hinaus innovative und bisher nicht oder nur teilweise in *Statechart*-Werkzeugen vorhandene intelligente Bearbeitungshilfen implementiert werden, welche teilweise [39] entnommen sind. Die vorliegende Anforderungsspezifikation umfasst dabei nicht sämtliche Funktionsmerkmale eines der in Abschnitt 3.1 vorgestellten *Statechart*-Editoren, sondern eine willkürliche Auswahl an Funktionalität aus verschiedenen *Statechart*-Editoren und zusätzlicher Funktionsmerkmale.

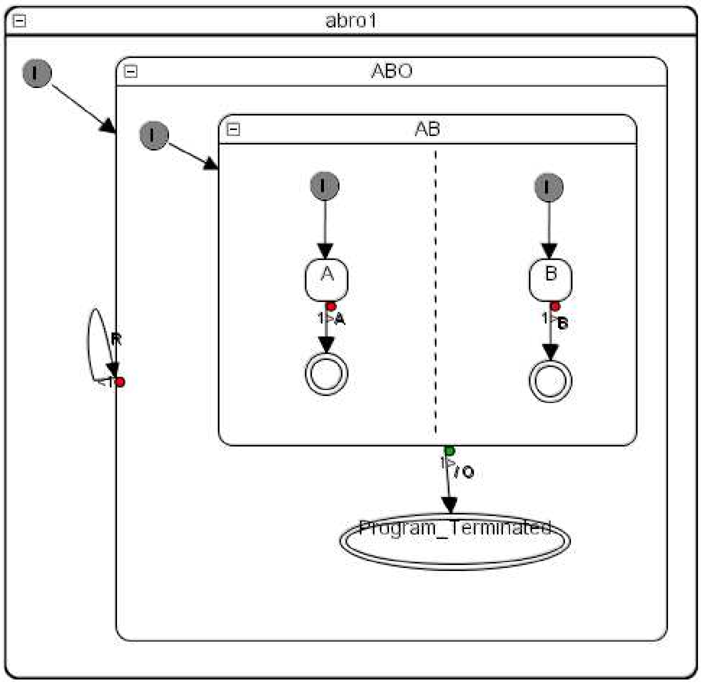


Abbildung 4.1.: Ein Referenzbeispiel

In diesem Kapitel werden die Anforderungen, welche an die vorliegende Arbeit gestellt werden, beschrieben. Neben der Umsetzung dieser Anforderungen wird darüber hinaus der Versuch unternommen, in Bezug auf Funktionalität und Er-

scheinungsbild eine kompetitive Software zu erstellen. Die hierfür gesteckten Ziele werden ebenfalls in diesem Kapitel vorgestellt und stellen eine Erweiterung der Anforderungen dar. Sämtliche Funktionsmerkmale, welche in diesem Kapitel dargestellt werden, sind im *KIEL*-Editor auch umgesetzt und können angewendet werden. Somit stellt das vorliegende Kapitel die Spezifikation des tatsächlichen Funktionsumfangs des Editors dar.

4.1. Syntaxgerichtetes Editieren

Unter Interpretation einer Zeichnung wird verstanden, den einzelnen Elementen der Zeichnung eine Bedeutung zuzuordnen. Wenn solch eine Zuordnung möglich ist, dann kann diese Zeichnung in eine andere Darstellungsform übersetzt werden, welche z. B. eine Baumstruktur oder eine Darstellung in Textform sein kann.

Grundsätzlich kann man graphische Editoren dahingehend unterteilen, in wieweit sie den Benutzer dazu verpflichten, bestimmte syntaktische Regeln bei der Eingabe einer Zeichnung einzuhalten, um damit eine Interpretation der Zeichnung zu ermöglichen. Kann der Benutzer beliebige Zeichnungen eingeben und sind beliebige Eingabeschritte erlaubt, so spricht man von freiem Editieren (engl. *Free-Hand-Editing*). Editoren jedoch, welche dem Benutzer eine definierte Menge an graphischen Formen anbieten und welche die Veränderung und Komposition dieser Formen nur in ganz bestimmter Art und Weise zulassen, definieren eine Menge von Eingaberegeln, welche die Syntax der eingebbaren Zeichnungen darstellen. Man spricht hier von syntaxgerichtetem Editieren (*Syntax-Directed-Editing*), also vom Einschränken der Eingabemöglichkeiten auf syntaxkonforme Varianten.

Es ist offensichtlich, dass syntaxgerichtetes Editieren in einem *Statechart*-Editor die Erstellung syntaktisch korrekter Diagramme vereinfacht. Grundsätzlich verringert syntaxgerichtetes Editieren die Menge an Benutzereingaben, die nötig sind, um ein gewünschtes Ergebnis zu erhalten. Wenn z. B. nur Rechtecke gezeichnet werden können, kann dem Benutzer das Wählen und Zeichnen der vier Kanten eines Rechtecks abgenommen werden und stattdessen das sprachspezifische Editierkommando *Rechteck zeichnen* (hier: *Zustand zeichnen*) angeboten werden.

Im Allgemeinen sollen bearbeitete *Statecharts* auch interpretiert werden können, um dann simuliert zu werden. Dies ist jedoch nur möglich, wenn der Aufbau des *Statecharts* den Regeln der Sprache genügt. Um den Benutzer von Eingaben abzuhalten, welche diesen Regeln zuwiderlaufen, werden solche Benutzeraktionen bei syntaxgerichtetem Editieren einfach ausgeschlossen. Hierbei werden all die Benutzeraktionen als regelkonform betrachtet, welche zu *Statecharts* führen, die ihrerseits ausschließlich durch Hinzufügen von weiteren Diagrammelementen zu vollständigen simulierfähigen Diagrammen erweitert werden können. Genau dieses Merkmal soll im vorliegenden *Statechart*-Editor umgesetzt werden.

In wieweit eine Eingabesyntax durch den Editor vorgegeben ist, bestimmt auch den Zeitpunkt, zu dem die eingegebene Zeichnung interpretiert werden kann. In Editoren, welche ausschließlich syntaxgerichtetes Editieren erlauben, kann nach je-

dem Eingabeschritt jedes Element der Zeichnung und somit die gesamte Zeichnung interpretiert werden, wenngleich sie noch nicht vollständig und damit korrekt sein muss. Jeder Eingabeschritt wirkt sich als Operation direkt auf das Datenmodell aus. Es ist daher möglich, das Datenmodell in einem Diagramm oder auch z. B. in einer Baumstruktur oder einer Repräsentation in Textform darzustellen. So können verschiedene Darstellungen, welche auf diesem Datenmodell beruhen, nach jedem Eingabeschritt aktualisiert werden.

Anders verhält es sich bei Editoren, welche freies Editieren erlauben. Bei solchen Editoren kann das Datenmodell im ungünstigsten Fall erst nach vollständiger Eingabe der Zeichnung erstellt werden. Erst dann nämlich können die gezeichneten Figuren (hier: Zustände und Transitionen) als vollständig angenommen werden und sodann diese Figuren, ihre Anordnung, Position und Größe, sowie ihre räumlichen Beziehungen wie Schachtelung (Hierarchie) und Berührung analysiert werden. Man spricht im Gegensatz zu semantischen Operationen hierbei von graphischem *Scanning*, einer lexikalischen Analyse, welche ihrerseits im günstigsten Falle zu einem korrekten *Statechart* führt [21].

Folgende Regeln sollen im *KIEL*-Editor umgesetzt werden:

- Geometrische Regeln:
  - Es können nur Rechtecke, Kreise und Pfeile eingegeben werden, wie sie in Abschnitt 2.3.2 angegeben sind. Es sollen genau die dort aufgezählten Elemente und diese nur als Ganzes erzeugbar sein.
  - Zwei Rechtecke oder Kreise dürfen sich nicht teilweise überlappen. Entweder ist das eine von beiden vollständig in dem anderen enthalten oder gar nicht.
  - Zustandskanten dürfen sich nicht beliebig nahe kommen (z. B. 10 Pixel Abstand, konfigurierbar).
  - Alle Diagrammelemente sind löschbar. Positionsänderbar sind alle Diagrammelemente mit Ausnahme der *Delimiter-Lines*. Größenänderbar sind alle Diagrammelemente außer *AND-States*.
  - Pfeile müssen als Anfangs- und Endpunkt die Kante eines Zustands besitzen.
- Semantische Regeln, siehe hierzu Abschnitt 2.3.2:
  - Nur in *Composite-States* können Zustände eingefügt werden.
  - Jeder *OR-State* und jede *Region* dürfen höchstens einen *Initial-State* besitzen.
  - *History-States* und *Deep-History-States* dürfen nur in *OR-States* und *AND-States* liegen und dies höchstens jeweils einmal.
  - *History-States* und *Deep-History-States* können nicht beliebig platziert werden. Sie 'sitzen' immer unter dem Expansionssymbol eines *Composite-States*.

- *Delimiter-Lines* dürfen nur im dafür vorgesehenen Bereich von *Composite-States* eingefügt werden, nicht jedoch im *Root-State*.
- *Initial-States* dürfen nur in *Regions* und *OR-States* liegen und dies höchstens jeweils einmal.
- Quellzustand und Zielzustand einer Transition müssen den gleichen Vater besitzen, d.h., Interlevel-Transitionen können nicht erzeugt werden.
- Transitionen dürfen nicht in *Initial-States*, *Suspends* und in den *Root-State* hineinführen.
- Transitionen dürfen nicht aus dem *Root-State*, *History-States*, *Deep-History-States* und *Final-States* hinausführen.
- Zustände dürfen nur dann als Endzustand deklariert werden, wenn sie keine ausgehenden Transitionen besitzen.
- Aus *Suspends* führen nur *Suspend-Transitions* heraus. Der Transitionstyp kann nicht geändert werden.
- Aus *Initial-States* führen nur *Initial-Arcs* heraus. Der Transitionstyp kann nicht geändert werden.
- Aus *Conditional-States* führen nur *Conditional-Transitions* heraus. Der Transitionstyp kann nicht geändert werden.

Diese Regeln werden angewendet, sobald der Benutzer ein neues Element hinzufügen möchte oder eine Verschiebe-, Größenänderungs- oder *Cut-and-Paste*-Aktion ausführt. Wird gegen eine der obigen Regeln verstoßen, so erscheint am Mauszeiger ein rotes Kreuz, siehe Abbildung 4.13(a), und die Aktion wird nicht ausgeführt. Eine Ausnahme hierzu bildet das automatische Layout in bestimmten Situationen, siehe Abschnitt 4.6.

Zum syntaxgerichteten Editieren gehört neben dem Erzeugen von Diagrammelementen im Falle von *Statecharts* auch die auf syntaktische Richtigkeit geprüfte Deklaration von Ereignissen, Variablen und Transitionsbeschriftungen. Es sollen eingebbar sein:

- *On-Entry-Actions*, *On-Inside-Actions* und *On-Exit-Actions* können für alle Zustände deklariert und bearbeitet werden.
- Lokale Ereignisse und lokale Variablen können für *Composite-States* deklariert werden.
- *Statechart*-globale *Input-Events* und *Output-Events* können in einem Dialogfenster editiert werden.
- Transitionsbeschriftungen können editiert werden im Format *Trigger {Condition} / Action*. Die in der Transitionsbeschriftung angegebenen Ereignisse und Variablen müssen vorher deklariert worden sein.

4.2. Graphische Darstellung eines *Statecharts*

Die graphische Repräsentation von *Statecharts* im *KIEL*-Editor soll der Spezifikation von *KIEL* entsprechen. Sie ist angegeben in den Abschnitten 2.3.2 und 2.3.3. Die Anzeige einiger dieser Elemente soll konfigurierbar sein, siehe hierzu Abschnitt 4.22.

4.3. Anwendung von verschiedenen Layouts

In *KIEL* werden *Statecharts* während der Simulation dynamisch visualisiert, d.h., die graphische Repräsentation eines *Statecharts* ändert sich mit jedem Simulationsschritt. Je nachdem, welche Zustände im aktuellen Schritt von Bedeutung sind, sind Zustände ein- oder ausgeklappt. Jede hieraus resultierende Kombination von ein- und ausgeklappten Zuständen wird Sicht (engl. *View*) des *Statecharts* genannt. Da ein eingeklappter Zustand weniger Platz als ein ausgeklappter Zustand einnimmt, erfordert jede Sicht ihr eigenes Layout. Das Layout jeder dieser Sichten entspricht jedoch unter Umständen – sofern es z. B. automatisiert erzeugt wird – nicht den Wünschen des Benutzers und bedarf deswegen einer Nachbesserung im Editor. Diese Änderungen sind jedoch nur dann sinnvoll, wenn sie gespeichert werden können, ansonsten sind sie nur im Browser während der Simulation verfügbar. Sofern in *KIEL* verschiedene Sichten eines *Statecharts* gespeichert werden können, soll der Editor verschiedene Sichten des *Statecharts* bearbeiten können.

4.4. Modellorientierung

Unter Modellorientierung können und sollen in diesem Kontext verschiedene Sachverhalte verstanden werden:

- *Statecharts* werden in *KIEL* in Form zweier Datenstrukturen repräsentiert, von denen die eine die hierarchische Struktur des *Statecharts* beschreibt und die andere Datenstruktur die graphische Repräsentation des *Statecharts* beinhaltet. Auf diesen Datenstrukturen operieren sämtliche *KIEL*-Module. Der *KIEL*-Editor soll aus Gründen der Interoperabilität auf der gleichen Struktur – auch Datenmodell genannt – arbeiten.

- Der *KIEL*-Editor soll den Entwurf verschiedener *Statechart*-Modelltypen unterstützen. Der Benutzer soll dabei zu Beginn des Editierens eines neuen *Statecharts* wählen können zwischen *Esterel-Studio*- und *Matlab*-Modellen und damit die Menge der zur Verfügung stehenden Diagrammelemente bestimmen können. Eine spätere Erweiterung um andere *Statechart*-Dialekte soll möglich sein, siehe hierzu im Anhang den Abschnitt A.3.3.

- Die Implementierung soll auf dem *MVC*-Muster (Abschnitt 2.4.3) basieren.

4.5. Verstecken innerer Zustände

Hierunter fällt die Veränderung der Sichtbarkeit der Unterzustände von Hierarchiezuständen.

Ähnlich wie beim Betrachten von *Statecharts* sind während des Bearbeitungsprozesses die Inhalte einiger Hierarchiezustände uninteressant und verbrauchen unnötig Darstellungsfläche. Aus diesem Grund soll es im Editor möglich sein, Hierarchiezustände zu kollabieren und zu expandieren. Dabei soll beim Einklappen neben dem Verstecken der inneren Zustände der Hierarchiezustand verkleinert werden sowie beim Ausklappen dessen Ursprungsgröße wiederhergestellt werden.

Zusätzlich zu diesen Anforderungen muss ein Mechanismus implementiert werden, welcher das Verschieben anliegender Zustände automatisch vornimmt, wenn für das Ausklappen nicht genügend Platz zur Verfügung steht, siehe hierzu Abschnitt 4.6. Des Weiteren soll das Ein- und Ausklappen als gleitende Strukturveränderung (Abschnitt 4.7) durchgeführt werden können .

4.6. Layout

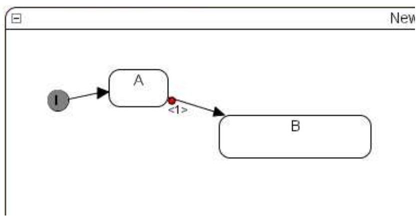
Grundsätzlich fördert ein übersichtliches Diagramm die Verständlichkeit desselben. Besonders bei großen Diagrammen bedeutet das manuelle übersichtliche Platzieren von Diagrammelementen einen großen Aufwand, welches dazu häufig noch iterativ durchgeführt werden muss. Während der Erweiterung eines *Statecharts* kann die Situation auftreten, dass das Gesamtbild für den Betrachter zunehmend schwerer verständlich wird, weil eine einheitliche Struktur nicht aufrechterhalten wird. Des Weiteren können sich im Verlauf der Bearbeitung Transitionen unvermeidbar überkreuzen, gleichartige Zustände unterschiedliche Größen besitzen und der zur Verfügung stehende freie Raum sich ungleichmäßig verteilen. Neben dem erhöhten Aufwand des manuellen iterativen Layoutens stellt sich als weiterer Nachteil heraus, dass ein nicht nur vom Kontext sondern auch vom Benutzer abhängiges Layout entsteht, was in Projekten zwecks angestrebter Einheitlichkeit der Diagrammdarstellung grundsätzlich unerwünscht ist. Dieses Problem löst die Layouter-Komponente von *KIEL*. Diese soll für das im Editor aktuell bearbeitete *Statechart* ein neues Layout berechnen. Das Ergebnis soll durch eine gleitende Strukturveränderung dargestellt werden, siehe Abschnitt 4.7.

Immer wieder bedarf es bei nicht vorausschauendem Entwurf des Diagramms des Schaffens von zusätzlichem Platz für neue Elemente. Daher soll zusätzlich zur Benutzung des *KIEL*-Layouters ein editoreigener Layoutmechanismus angewendet werden, wenn für einen neu einzufügenden Zustand nicht genug Platz zur Verfügung steht, siehe Abschnitt 4.13.

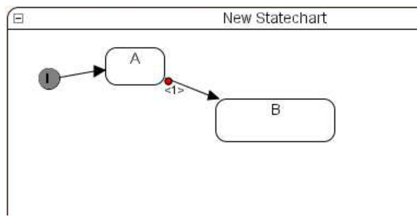
4.7. Layoutgestützte gleitende Strukturveränderung

Mit Struktur ist hier die graphische Anordnung der Zustandsdiagrammelemente gemeint. Unter Veränderung dieser Anordnung versteht man die Änderung der Position oder Größe von Zustandsdiagrammelementen. Eine Strukturveränderung (engl. *Morphing*) wird gleitend genannt, wenn die Veränderung nicht in einem Schritt sondern als Folge von sich geringfügig unterscheidenden Anordnungen durchgeführt wird. Für den Betrachter soll aus der Ausgangsanordnung heraus die Zielanordnung erwachsen, siehe Abbildung 4.2.

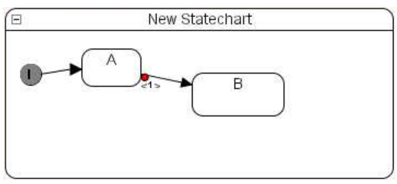
Worin liegt der Sinn einer gleitenden Veränderung? Große Veränderungen erschweren es dem Benutzer, diese nachzuvollziehen. Der Überblick kann verloren gehen. Daneben könne sie zu einer Verringerung der Akzeptanz der programmgesteuerten Layoutveränderungen führen. Der Benutzer entwirft ein *Statechart* derart, wie er sich die *Statechart*-Struktur gedanklich vorstellt. Dies bezeichnet man auch als *Mental-Map*. Dabei ist die vorgestellte Struktur die im ersten Moment einzig korrekte Struktur aus Sicht des Benutzers. Jedes hiervon abweichende Layout muss daher als Vorschlag dem Benutzer gegenüber dargebracht werden.



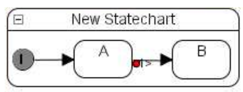
(a) Anfangsanordung



(b) 1. Zwischenschritt



(c) 2. Zwischenschritt



(d) Zielanordnung

Abbildung 4.2.: Layoutgestützte gleitende Strukturveränderung

Im hier zu implementierenden Editor soll eine gleitende Strukturveränderung bei Ausführung eines Layoutvorgangs (Abschnitt 4.6), beim Kollabieren und Expandieren von Zuständen (Abschnitt 4.5), sowie beim automatisch durchgeführten Vergrößern von Zustandsdiagrammelementen als Folge von Platzproblemen beim Positionieren von neuen Elementen (Abschnitt 4.13) durchgeführt werden.

4.8. Rasterorientiertes Bearbeiten

Im Editor werden alle graphischen Elemente in einem Koordinatensystem angeordnet. Die linke obere Ecke hat die Koordinaten (0,0). Dieses ganzzahlige Koordinatensystem kann man als Raster auffassen mit Rastergröße 1.

Im Falle eines *Statechart*-Editors ist eine so feine Rasterung jedoch nicht notwendig und erschwert sogar das Anordnen von Elementen, sofern sie auf eine bestimmte Art und Weise relativ zueinander angeordnet werden sollen wie z. B. die Ausrichtung an einer Geraden.

Eine Lösung dieses Problems bietet die Veränderung der Rastergröße auf z. B. ein Viertel der Größe von *Pseudo-States*, also etwa 5 Pixel. In diesem Fall wären die Koordinaten aller Diagrammelemente ganzzahlige Vielfache von 5. Will man bei solch einem Raster ein Element verschieben, so springt es zwischen den möglichen Koordinaten. Setzt man das Element ab, hat es Koordinaten mit der genannten Eigenschaft. Dies versteht man unter rasterorientiertem Bearbeiten (engl. *Snapping*). Das Raster soll in Form von Punkten sichtbar sein, siehe Abbildung 4.3.

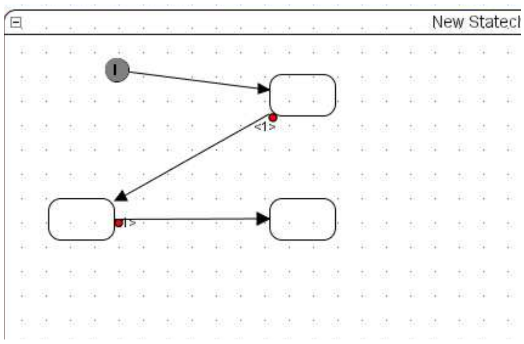


Abbildung 4.3.: Anzeige des Bearbeitungsrasters

Der Editor soll eine vom Benutzer wählbare Rastergröße besitzen und gleichzeitig das Raster in Form von Punkten auf der Zeichenfläche darstellen, siehe hierzu Abbildung 4.3. Die Konfigurationsmöglichkeiten werden im Anhang im Abschnitt .2.1 beschrieben.

4.9. Nachvollziehbarkeit von Bearbeitungsschritten

Ein Bearbeitungsschritt ist hier die Einheit, auf der ein gewöhnlicher *Undo-and-Redo*-Mechanismus basiert (siehe auch Abschnitt 4.14). Die Anzahl der verfügbaren *Undo*- und *Redo*-Schritte soll in der Statusleiste dargestellt werden und es soll möglich sein, in einer Historie die einzelnen Schritte mit einem Zeitstempel versehen chronologisch aufzulisten (siehe hierzu im Anhang den Abschnitt .1.1). Darüber hinaus soll diese Historie in dem in Abbildung 4.4 dargestellten Format exportierbar sein.

```
13:31:29;Esterel Studio;Start
13:31:32;Simple State;Start
13:31:32;Simple State;Successful
13:31:36;Initial State;Start
13:31:38;Initial State;Successful
13:31:41;Trace Log;Start
```

Abbildung 4.4.: Ein *Trace-Log* von drei Benutzeraktionen



(a) Vor Beginn eines Abspielvorgangs



(b) Während eines Abspielvorgangs

Abbildung 4.5.: Die Werkzeugleiste zum Abspielen von Bearbeitungsschritten

In komplexen Diagrammen kann die Frage auftreten, welcher Bearbeitungsweg zu dem vorliegenden Resultat geführt hat. Der Benutzer kann nun mehrfach einen *Undo*-Schritt ausführen und hiernach Schritt für Schritt ein *Redo*, um die Entwicklung des Diagramms über die letzten Bearbeitungsschritte anzusehen. Zur Vermeidung einer hohen Mausklick-Rate zur Durchführung dieses Vorhabens soll dem Benutzer ein Wiedergabe-Knopf zur Verfügung gestellt werden, wie er auch bei z. B. Tonträger-Abspielgeräten angeboten wird, neben einer Pause-Taste, schnellem Vorlauf und Rücklauf, sowie schrittweisem Vor- und Rücklauf, siehe Abbildung 4.5. Wiedergabe bedeutet hier, dass zunächst sämtliche *Undo*-Schritte ausgeführt werden und danach sukzessive die *Redo*-Schritte. Die Stop-Taste beendet den Wiedergabe-Modus und führt wieder zum Ausgangspunkt. Zusätzlich wird dem Anwender, wie er es aus Media-Player-Software gewohnt ist, ein Schieberegler zur Schrittanwahl an die Hand gegeben. Der Benutzer kann darüber hinaus an jeder Stelle des Wiedergabe-Modus diesen mit der Pause-Taste beenden und neue Bearbeitungsschritte ausführen.

4.10. Mehrschichtige Übersichtsdarstellungen

Der Benutzer soll frei wählen können, wie viel Information er gleichzeitig betrachten möchte. Idealerweise ist der editierbare Bereich der gesamte Bildschirm. Werden Übersichten nicht schichtweise angezeigt, sondern nebeneinander, so geht Platz für die Bearbeitung des Diagramms verloren. Daher soll die Möglichkeit bestehen, auf verschiedenen durchsichtigen Schichten *Statechart*-Informationen darzustellen (engl. *Layering*), z. B. eine nicht editierbare *Statechart*-Übersicht unter einem mo-

mentan bearbeiteten *Statechart*-Ausschnitt. Des Weiteren soll ein Diagramm allgemein aus mehreren Ebenen bestehen können, welche unterschiedliche Ausschnitte des Diagramms zeigen [9].

Übersichtsdarstellungen zeigen das Diagramm in der Regel vollständig. Denkbar ist allerdings auch die Darstellung eines Diagrammteils. Unterschiedliche Darstellungen für den Inhalt des Diagramms sind dabei vorstellbar. Ein *Statechart* kann neben der Diagrammdarstellung auch als Baum oder in Textform wiedergegeben werden. Fasst man die Darstellungen als durchsichtige Bilder auf, so spricht man von der Darstellung mehrerer überlagerter Ebenen oder von Mehrschichtigkeit. Beispielsweise kann das bearbeitbare Diagramm im Ausschnittsformat als oberste Schicht angezeigt werden, wobei darunter, in leichteren Farbtönen wie Grau, das gesamte Diagramm zur Übersicht dargestellt wird. Zur besseren Orientierung soll das aktuell selektierte Diagrammelement auch in der Übersichtsdarstellung hervorgehoben werden. Abbildung 4.6 zeigt neben der Bearbeitungsschicht eine Übersichtsschicht. In der Abbildung ist die Hervorhebung des selektierten Diagrammelements in der Übersichtsansicht kaum sichtbar.

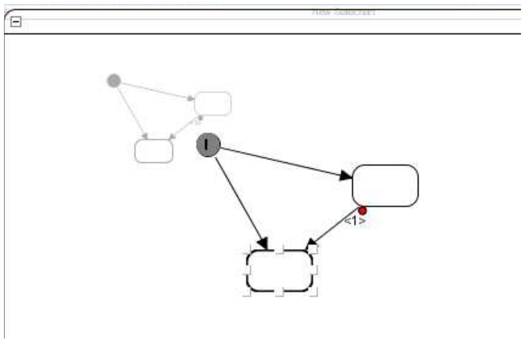


Abbildung 4.6.: Bearbeitungsschicht und Übersichtsschicht

Mehrschichtige Darstellungen können den Benutzer unter Umständen irritieren oder sogar überfordern. Mit solchen Darstellungen soll daher zum einen sparsam umgegangen werden und sie sollen ausschaltbar sein, und zum anderen soll viel Wert auf die Auswahl der Farben gelegt werden. Beispielsweise ist es sinnvoll, im Mehrschicht-Anzeige-Modus das aktuell bearbeitete Diagrammelement in allen Schichten farblich leicht hervorzuheben.

Im Allgemeinen sind Übersichten nicht bearbeitbar. Denkbar wäre allerdings in diesem Kontext, alle angezeigten Darstellungen zum Editieren freizugeben und die Reihenfolge der Schichten änderbar zu gestalten. Im vorliegenden Editor sollen zwei Darstellungen übereinander gelegt werden: Die Bearbeitungsschicht als oberste Schicht liegt über der nicht editierbaren Übersichtsschicht, welche farblich ausgegraut ist.

4.11. Farbliche Syntaxprüfung

Wie im Abschnitt über syntaxgerichtetes Editieren erwähnt, sollen im Editor modellierte *Statecharts* konform zu der von einem *Statechart*-Simulator geforderten Syntax sein. Das syntaxgerichtete Editieren (Abschnitt 4.1) stellt zwar sicher, dass während des Bearbeitungsvorgangs Eingaben, die gegen syntaktische Regeln verstoßen, nicht zugelassen werden. Dies führt allerdings nicht zu jederzeit syntaktisch korrekten Diagrammen, da auch unvollständige Diagramme akzeptiert werden müssen, sofern sie durch Hinzufügen weiterer Elemente zu einem syntaktisch korrekten Diagramm führen können.

Der Benutzer soll bei Beurteilung der Frage unterstützt werden, ob alle vorliegende Diagrammelemente vollständig sind im Sinne der zu prüfenden Syntax. Ist dies für ein Element nicht der Fall, so wird das Element rot gezeichnet. Ein Diagrammelement ist vollständig, wenn folgende Regeln eingehalten werden:

Das Element ist

- ein *OR-State* aber nicht der *Root-State* und enthält einen *Initial-State* und hat eingehende Transitionen.
- der *Root-State* und enthält einen *Initial-State*.
- ein *AND-State* und enthält eingehende Transitionen.
- eine *Region* und enthält einen *Initial-State*.
- ein *Simple-State* und enthält eingehende Transitionen.
- ein *Initial-State* und enthält ausgehende Transitionen.

Diese Regeln werden unabhängig vom gewählten Modell geprüft, d.h., für Matlab-Modelle und *Esterel-Studio*-Modelle werden die gleichen Regeln angewendet.

4.12. Positionierungshilfen und Ausrichtungshilfen

Ein wesentlicher und immer wiederkehrender Bearbeitungsschritt ist der des Positionierens von Diagrammelementen. Elemente sollen übersichtlich angeordnet sein. So werden sie entweder relativ zueinander platziert oder mit absoluten Koordinaten versehen. Zur Unterstützung der absoluten Positionierung werden in der Statusleiste des Editors die Mauszeigerkoordinaten angezeigt. Sinnvoll wäre auch, die gewünschten Koordinaten eines Elements direkt eingeben zu können.

Der Benutzer soll Elemente relativ zueinander anordnen können, indem zunächst mehrere Elemente mit der *CTRL*-Taste selektiert werden und sodann ausgewählt wird, wie alle selektierten Elemente relativ zum ersten selektierten Element angeordnet werden sollen: links-, rechts-, oben- oder untenbündig, gleiche Breite oder Höhe, vertikal oder horizontal zentriert.

4.13. Layouthilfen während der Bearbeitung

Eine Layoutmaßnahme kann nicht nur nach einem Bearbeitungsschritt notwendig werden sondern auch während eines Bearbeitungsschrittes. Dies ist z. B. der Fall, wenn der Benutzer einen Zustand an einer Stelle hinzufügen möchte, an der nicht genug Platz ist, die Zustandskanten also die eines anderen Zustands kreuzen würden. Dann ist diese Hinzufüge-Operation an dieser Stelle nicht erlaubt. Folglich müsste abgebrochen werden, der Benutzer müsste durch eine geeignete Layoutmaßnahme Platz schaffen und könnte danach erneut versuchen, die Hinzufüge-Operation an der gewünschten Stelle durchzuführen.

Schon allein *das Platz-Schaffen* kann eine aufwändige Maßnahme sein. Man stelle sich vor, der Vater des neuen Zustands würde durch eine notwendige Vergrößerung seinen eigenen Vater schneiden. Dieses *Platz-Schaffen* soll nun im *KIEL*-Editor durch einen geeigneten Algorithmus automatisiert werden, indem diese Operation vom Editor ausgeführt wird, wann immer dieser die Notwendigkeit solch einer Layoutmaßnahme erkennt, nämlich genau während des Platzierens eines neuen Zustands. Während eines Bearbeitungsschritts wird also eine Layouthilfe seitens des Editors angeboten und durchgeführt.

4.14. *Undo-and-Redo*

Diese aus Editoren allgemein bekannten Funktionen bewirken ein Zurücknehmen (engl. *Undo*) beziehungsweise ein Zurücknehmen des Zurücknehmens (engl. *Redo*) eines Bearbeitungsschrittes. Die *Undo*-Funktion dient der Korrektur ungewollter Eingaben. Stellt sich unmittelbar nach Anwendung der *Undo*-Funktion heraus, dass doch keine ungewollte Eingabe vorlag, so kann der *Undo*-Schritt rückgängig gemacht werden mit der *Redo*-Funktion. *Redo*-Schritte sind nur im direkten Anschluss an *Undo*-Schritte möglich. Sobald nach *Undo*-Schritten ein neuer Bearbeitungsschritt ausgeführt wird, sind keine *Redo*-Schritte mehr verfügbar.

4.15. *Cut-and-Paste*

Auch hierbei handelt es sich um eine aus Editoren allgemein bekannte Funktion. Ein Diagrammelement kann ausgeschnitten werden (engl. *Cut*), und nach Belieben häufig wieder eingefügt werden (engl. *Paste*), und das in beliebigen Anwendungen, die *Cut-and-Paste* unterstützen und die mit den ausgeschnittenen Daten etwas anfangen können. Im vorliegenden Editor soll eine Variante implementiert werden: Ein ausgeschnittenes Diagrammelement soll höchstens ein Mal und nur im Editor wieder eingefügt werden. So ist ein Duplizieren von Elementen ausgeschlossen.

4.16. Zoom und Ausschnittsänderung

Die Ansicht des vorliegenden Diagramms soll nach Belieben vergrößert und verkleinert werden können. Hierfür soll zum einen der Zoomfaktor gleitend über einen Schieberegler in der Statusleiste eingestellt werden können, siehe Abbildung 4.7, zum anderen sollen über das Menü einzelne *Zoom-In*- und *Zoom-Out*-Schritte ausgeführt werden können. Des Weiteren soll mittels *CTRL* + *Plus*-Taste und *Minus*-Taste sowie mit Hilfe *CTRL* + Mausrad gezoomt werden können.

Der sichtbare Bereich des *Statecharts* soll über Scrollleisten, das Mausrad sowie über die *Cursor*-Tasten geändert werden können (engl. *Panning*).



Abbildung 4.7.: Zoomregler zur gleitenden Einstellung des Zoomfaktors

Zum anderen soll schrittweise gezoomt werden können mit *Zoom-In*- und *Zoom-Out*-Buttons. Die Zoom-Schrittweise soll darüberhinaus konfigurierbar sein, siehe im Anhang den Abschnitt .2.1.

4.17. Diagrammelementtyp-Morphing

Hierunter wird das Ändern des Typs eines Diagrammelementes verstanden. Während des Bearbeitungsprozesses kann die Situation auftreten, dass ein Diagrammelement gegen ein anderes ausgetauscht werden muss, z. B. ein *Simple-State* gegen einen *OR-State*, weil in diesem *Simple-State* weitere Zustände platziert werden sollen. Eigentlich müßte jedoch lediglich der Diagrammelementtyp geändert werden. Das Ersetzen dieses Zustands gegen einen neuen Zustand hätte zur Folge, dass alle mit dem zu ersetzenden Zustand verbundenen Transitionen und sonstigen Eigenschaften verloren gingen. Daher soll der Editor Möglichkeiten bieten, den Diagrammelementtyp zu ändern, sofern dies syntaktisch gesehen möglich ist. Ein beliebiger Zustand soll z. B. in einen Endzustand überführt werden können, wenn er keine ausgehenden Transitionen besitzt. Ein *OR-State* hingegen soll den Typ eines *Simple-States* erhalten können, wenn dieser *OR-State* keine Zustände beinhaltet.

4.18. Änderung von Transitionsprioritäten

Im *KIEL*-Editor sollen Transitionsprioritäten vergeben, angezeigt und geändert werden können, siehe auch Abschnitt 2.3.3. Beim Erzeugen einer neuen Transition soll der Editor automatisch dieser Transition eine Priorität zuweisen, wobei keine zwei Transitionen die gleiche Priorität haben dürfen. Angezeigt sollen Prioritäten

werden in spitzen Klammern an der Transition in der Nähe des Quellzustands. Der Benutzer soll eine Priorität ändern können, indem er eine Transition anwählt und ihr eine neue Priorität p zuweist. Hiernach sollen für alle Transitionen die Prioritäten neu berechnet werden derart, dass die ursprüngliche Reihenfolge erhalten bleibt, die Transition mit p aber in der Reihenfolge neu eingeordnet wird.

4.19. Ansprechendes Benutzeroberflächendesign

Das *KIEL*-Projekt soll nicht nur professionellen Anforderungen bezüglich seiner Funktionalität genügen, es soll daneben vom äußeren Erscheinungsbild mit kommerziellen Produkten konkurrieren können. Hierfür ist die Benutzeroberfläche von zentraler Bedeutung. Die Anwendung muss eine Freude für die Augen sein. Daher ist eine ansprechende farbliche Gestaltung vorteilhaft. Die gewählten Farbverläufe sollen denen neuester Office-Produkte gleichen. Daneben ist das weite Feld der Software-Ergonomie zu berücksichtigen. Folgende Aspekte sollen bei der Implementierung umgesetzt werden:

- Werkzeugleisten für die komfortable Anwendung häufig genutzter Funktionen, siehe Abbildung 4.8,
- Ausblendbarkeit von Menüs und der Statusleiste, siehe Abbildung 4.8, sowie der Baumansicht,

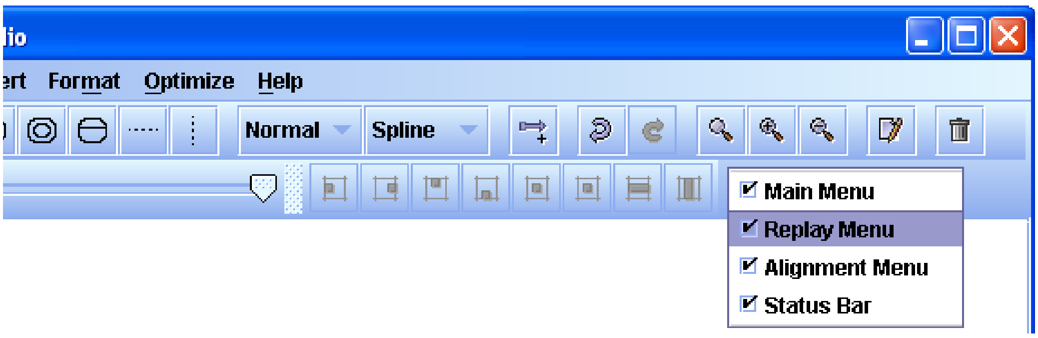
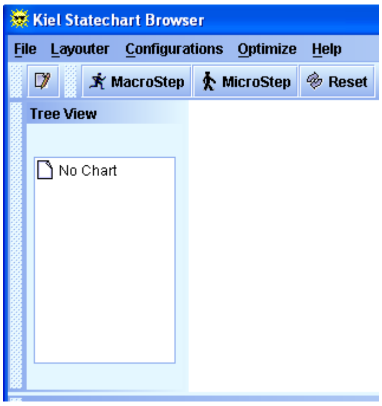
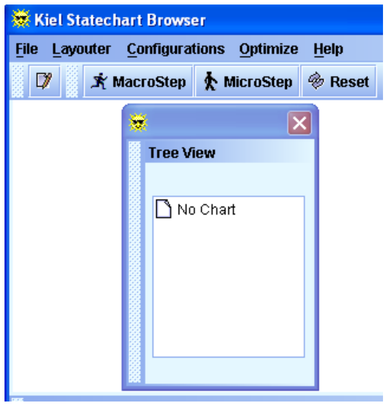


Abbildung 4.8.: Ausblendbarkeit von Werkzeugleisten-Elementen

- fließende Komponenten, die per *Drag-and-Drop* aus dem *KIEL*-Anwendungsfenster gezogen werden können
- *Popup*-Menüs, welche abhängig vom Typ des aktuell selektierten Diagrammelementes zusammengestellt werden, siehe Abbildungen 4.10 und 4.11.



(a) Eingebettete Komponente der Benutzeroberfläche



(b) Fließende Komponente der Benutzeroberfläche

Abbildung 4.9.: Fließende Benutzeroberflächen-Komponenten

- *Shortcuts* und *Icons* für Menüs und Menü-Elemente
- sinnvolle Gruppierung von Menü-Elementen, so dass ungewünschte Funktionen nicht versehentlich anwählbar sind,
- Bestätigungsdialoge bei nicht widerrufbaren Operationen,
- Mauszeiger mit visuellen Hinweisen auf den aktuellen Bearbeitungsschritt,
- Eine Dateihistorie zum schnellen Zugriff auf kürzlich geladene und gespeicherte Diagramme sowie
- die Wählbarkeit der in der Benutzeroberfläche verwendeten Sprache (wahlweise Englisch oder Deutsch). Siehe hierzu Abbildung 4.12.

4.20. *KIEL*-Anwendungsfenster

Neben dem Editor-Modul soll eine Rahmenanwendung in Form eines Anwendungsfensters entwickelt werden, in welches das zu entwickelnde Editor-Modul und das Browser-Modul eingebettet werden können und welches als Kommunikationsmittler zwischen beiden Modulen agiert. Wie in Abschnitt 5.2.1 beschrieben, sollen die *KIEL*-Module einerseits weitgehend unabhängig voneinenander implementiert sein, andererseits sollen jedoch der Editor und der Browser das aktuell angezeigte *Statechart* untereinander austauschen können.

Das Anwendungsfenster soll darüber hinaus für jedes der beiden Module folgende fünf unterschiedliche Komponenten einbetten, von denen jede der ersten drei Komponenten als fließende Komponente (engl. *Floatable-Components*) aus dem

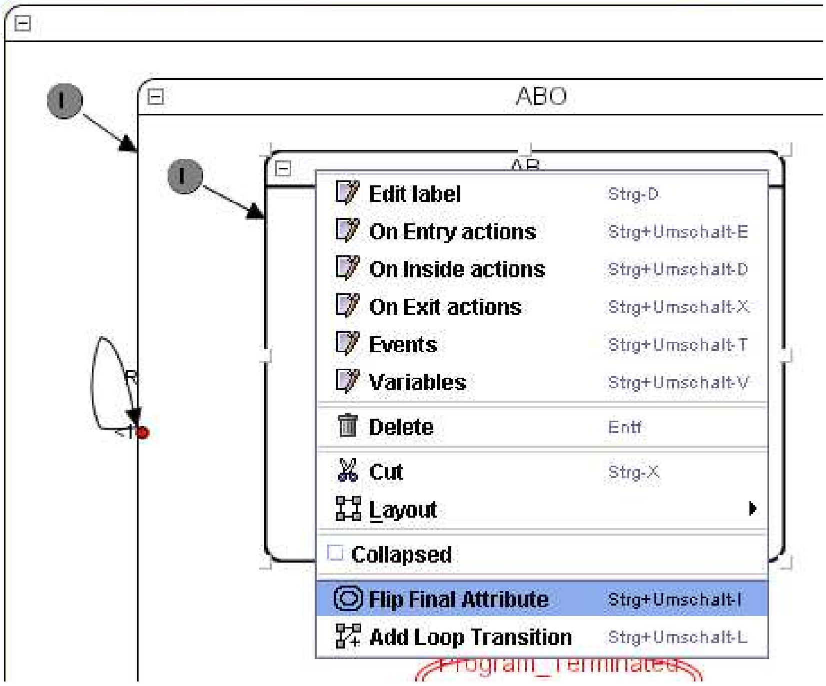


Abbildung 4.10.: Kontext-Menü zu einem *OR-State* als *Popup*-Menü

Anwendungsfenster herausgezogen werden kann und dann in einem eigenständigen Fenster solange existiert, bis dieses wieder geschlossen wird und sodann seinen Platz im Anwendungsfenster zurück erhält, siehe Abbildung 4.9:

- eine Baumansicht,
- eine Register-Ansicht mit beliebig vielen Registern,
- eine Werkzeugleiste,
- die Hauptkomponente, dies ist die graphische Anzeige des aktuellen *Statecharts* und
- Menüs im Anwendungsfenster-Hauptmenü.

Folgende Funktionen soll das Anwendungsfenster bereitstellen:

- Dateien öffnen: Das *File-Interface*-Modul von *KIEL* soll angebunden werden.
- Dateinamen speichern: Speichern der Namen der zuletzt geöffneten Dateien soll angeboten werden.
- Nachrichtendialog unterdrücken: Unerwünschter Meldungen in Meldungsdialogfenstern sollen unterdrückbar sein.

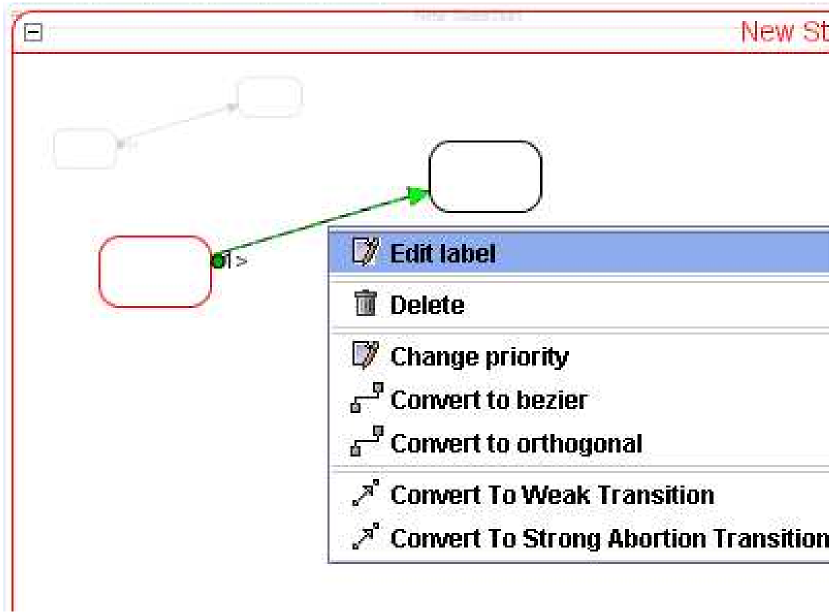
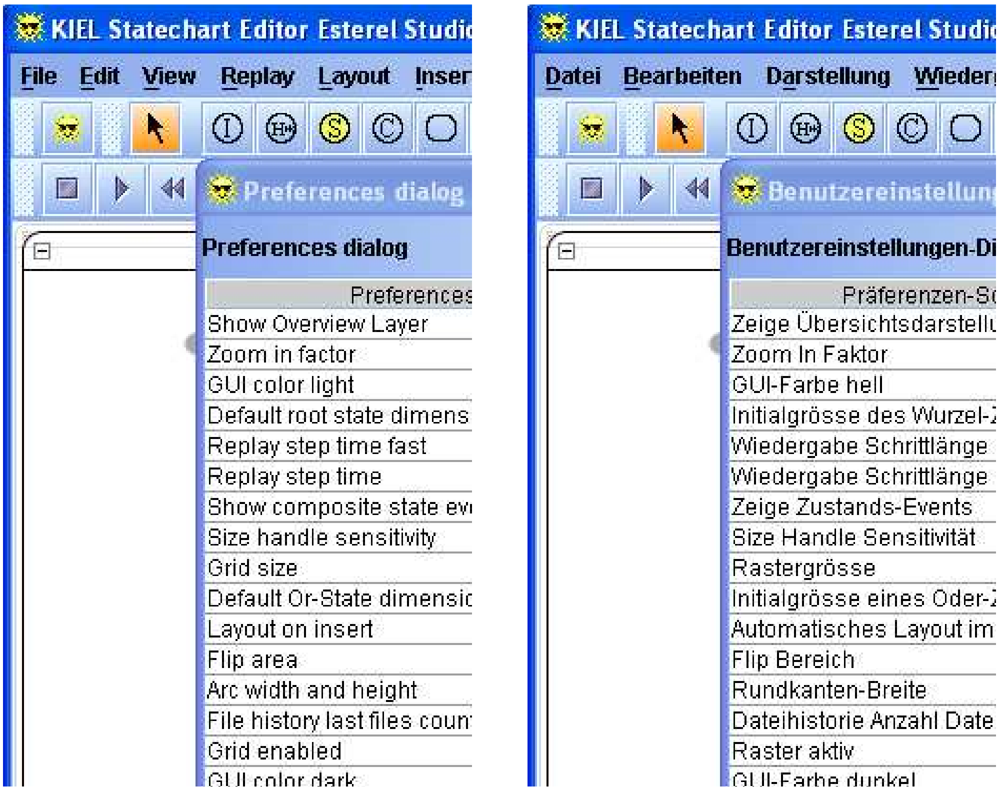


Abbildung 4.11.: Kontext-Menü zu einer Transition als *Popup*-Menü

- Anwendung beenden: Eine Sicherheitsabfrage zum Speichern vor Beendigung der Anwendung soll angezeigt werden.
- Menü-Elemente konfigurieren: Die Sichtbarkeit von Menüs soll einstellbar sein.
- Meldungen anzeigen: Fehlermeldungen und Hinweise sollen in der Statusleiste dargestellt werden können.
- zwischen Editor und Browser umschalten: Das aktuelle *Statechart* soll bei Wechsel von Editor-Anzeige nach Browser-Anzeige zwischen beiden Komponenten ausgetauscht werden und gleichzeitig soll die exklusive Sichtbarkeit beider Komponenten sichergestellt werden.

4.21. Intuitive Bedienbarkeit

Der Editor soll intuitiv bedienbar sein, das heißt, die Software muss sich zum einen so verhalten, wie es der Anwender erwartet, zum anderen müssen alle Funktionalitäten leicht aufrufbar sein. Hierfür ist notwendig, dass Funktionsweise und Erscheinungsbild der graphischen Benutzeroberfläche anderen Editoren und insbesondere *Statechart*-Editoren ähnelt. Als Vorbild für den *KIEL*-Editor soll das Werkzeug *Esterel-Studio* (Abschnitt 3.1.2) gewählt werden. Der Grad an intuitiver Bedienbarkeit eines Diagrammeditors lässt sich an der Art und Weise messen, wie sich:



(a) Sprachliche Anzeigeelemente auf englisch (b) Sprachliche Anzeigeelemente auf deutsch

Abbildung 4.12.: Mehrsprachigkeit

- neue Diagrammelemente erzeugen lassen (siehe Abschnitt 4.21.1),
- Selektieren, Verschieben, Vergrößern und Verkleinern von Diagrammelementen realisieren lässt (siehe Abschnitt 4.21.2) und
- syntaxgerichtetes Editieren (Abschnitt 4.1) auf die Arbeit mit dem Editor auswirkt.

4.21.1. Erzeugung neuer Diagrammelemente

Esterel-Studio kennt wie andere Editoren auch das Konzept der Bearbeitungs-Modi. Hier beschreibt ein Modus die Art und Weise, in welcher auf Benutzereingaben reagiert wird. Grundsätzlich besitzt jede Software mindestens einen Modus. Sobald jedoch Benutzeraktionen möglich sein sollen, die in einem bestimmten Kontext ausgeführt werden, die also z. B. sich auf vorhergehende Benutzeraktionen beziehen oder von diesen abhängen, muss auch für diese Aktionen ein Modus definiert werden. Ein Modus zeichnet sich aus durch das Vorhandensein einer Transaktionsklammer, innerhalb der mehrere Benutzeraktionen entweder ganz oder überhaupt

nicht ausgeführt werden.

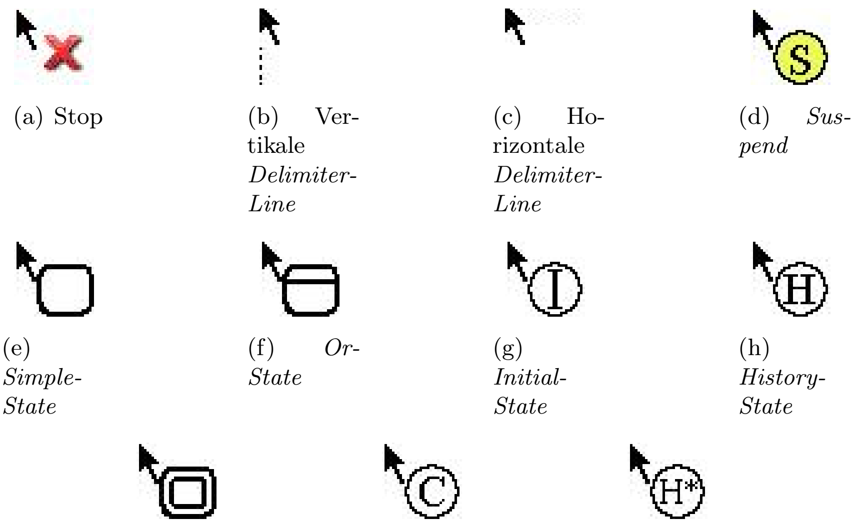
Ein *Statechart*-Editor soll sich in Abhängigkeit des jeweiligen Anwendungsfalls verhalten. Während ein neuer Zustand platziert wird, wird der Editor ein anderes Verhalten aufweisen müssen, als wenn eine Transition gezeichnet wird. Diese beiden Anwendungsfälle bestehen aus mehreren Schritten. Zunächst wird der Mauszeiger platziert und nach dem Mausklick wird der Zustand oder die Transition erzeugt. Wie sich hierbei der Mauszeiger verhält und welche Positionsüberprüfungen durchgeführt werden, hängt davon ab, ob ein Zustand oder eine Transition erzeugt werden soll. Daher muss der Benutzer zunächst festlegen, was er erzeugen will, damit im nächsten Schritt seine Mausbewegungen entsprechend interpretiert werden können.

Dieser Sachverhalt gilt für alle Editoren. Daher wird das Konzept der Bearbeitungs-Modi vielfach angewendet. Der Standard-Modus ist der Selektionsmodus. In den Transitionsmodus soll der Benutzer gelangen, sobald sich der Mauszeiger in einer Position befindet, in der durch Mausklick das Zeichnen einer Transition begonnen werden könnte. Daneben gibt es für alle hinzufügbaren Diagrammelemente jeweils einen weiteren Modus, da beim Bewegen des Mauszeigers geprüft werden muss, ob das Erzeugen des Diagrammelementes an der aktuellen Stelle erlaubt ist und die hierfür notwendige Überprüfungsregel von der Art des hinzuzufügenden Diagrammelementes abhängig ist.

Abbildung 4.13 zeigt die Palette an Mauszeigern, welche der Editor verwenden soll, um anzuzeigen, welcher Bearbeitungsschritt gerade ausgeführt wird. Wählt der Benutzer z. B. das Hinzufügen eines *Simple-States* in der Werkzeugleiste, so verwandelt sich der Mauszeiger auf der Zeichenfläche überall dort in den in Abbildung 4.13(e) dargestellten Mauszeiger, wo dieses Diagrammelement platziert werden darf. Eine besondere Bedeutung besitzt der Mauszeiger in Abbildung 4.13(a). Dieser Mauszeiger erscheint immer dann, wenn die gewählte Aktion an der aktuellen Position des Mauszeigers nicht möglich ist.

4.21.2. Selektion und Manipulation von Diagrammelementen

Wenn der Benutzer nicht das Erzeugen eines Diagrammelements angekündigt hat durch Wahl eines entsprechenden Modus, befindet sich der Editor im Selektionsmodus, in welchem mittels des Mauszeigers Diagrammelemente verschoben, verkleinert, vergrößert und gelöscht werden können. Der Benutzer ist gewohnt, dass immer dasjenige Diagrammelement selektiert ist, auf dem sich der Mauszeiger gerade befindet. Sofern Elemente ineinander geschachtelt sind, erwartet der Benutzer, dass das kleinste Element selektiert wird. Die Selektion wird häufig durch die Farbgebung, die Linienstärke oder das Erscheinen so genannter *Size-Handles* angezeigt. Die Visualisierung der Selektion in anderen Ansichten wird in Abschnitt 4.10 dargestellt.



(a) Stop (b) Vertikale *Delimiter-Line* (c) Horizontale *Delimiter-Line* (d) *Suspend*

(e) *Simple-State* (f) *Or-State* (g) *Initial-State* (h) *History-State*

(i) *Final-Simple-State* (j) *Choice* (k) *Deep-History-State*

Abbildung 4.13.: Verschiedene Mauszeiger

4.22. Konfigurierbarkeit

Der Editor soll auf die individuellen Bedürfnisse des Benutzers eingestellt werden können. Die Konfiguration soll über ein Dialogfenster erfolgen können, siehe hierzu Abbildung .2 und die Einstellungen sollen wahlweise gespeichert werden können, so dass sie bei erneutem Start der Anwendung nicht erneut eingegeben werden können. Wünschenswert wäre ein sofortiges Aktivwerden der vom Benutzer getätigten Einstellungen. Ungültige Werte werden automatisch verworfen und durch den letzten gültigen eingegebenen Wert oder den Standardwert ersetzt. Zu jeder Einstellungsmöglichkeit soll ein Standardwert vorgegeben sein, damit der Benutzer schon nach dem ersten Start der Anwendung sinnvoll arbeiten kann. Die Konfigurationsmöglichkeiten und die hierzu definierten Standardwerte sind im Anhang im Abschnitt .2.1 angegeben. Neben dieser Art der Konfiguration über ein Dialogfenster und wahlweiser Speicherung soll die Anzeige der folgenden Editor-Elemente nur temporär und über das Menü `View` erreichbar ein- und ausgestellt werden können:

- Übersichtsdarstellung,
- Raster,
- *Actions* eines Zustands,
- lokale *Events* eines Zustands und
- lokale Variablen eines Zustands.

4.23. Bearbeitungsschritt-Historie

Es sollen zwei Historien der vom Benutzer durchgeführten Bearbeitungsschritte gepflegt werden. In einer Historie sollen sämtliche Schritte angezeigt werden, wie z. B. der Beginn des Anlegens eines Diagrammelements, das erfolgreiche oder erfolglose Abschließen dieses Schrittes, das Öffnen des *Preferences*-Dialogs und andere. In einer zweiten Historie sollen nur diejenigen Schritte dargestellt werden, auf welche der *Undo-and-Redo*-Mechanismus anwendbar ist. In beiden Historien sollen die Zeitpunkte angegeben sein, es soll die Anzeigesprache (deutsch / englisch) gewählt werden können und die Daten sollen exportierbar sein. Da es sich um Tabellenstrukturen handelt, werden die Daten zeichensepariert exportiert im CSV-Format (engl. *Character-Separated-Values*), wobei das Trennzeichen frei wählbar sein soll. Abbildung .4 zeigt links die *Undo-and-Redo*-Historie, in der farblich unterschieden wird zwischen Schritten, auf welche der *Undo*-Mechanismus angewendet werden kann und Schritten, für welche der *Redo*-Mechanismus zur Verfügung steht.

5. Umsetzung und Implementierung

Nachdem im vorangegangenen Kapitel der Funktionsumfang des *KIEL*-Editors spezifiziert wurde, wird in diesem Kapitel nun unter Beachtung dieser Spezifikation zunächst die Vorgehensweise (5.1) für die Umsetzung und Implementierung des *KIEL*-Editors festgelegt. Hiernach werden in den Abschnitten 5.2 und 5.3 die Ergebnisse der iterativ-inkrementellen Design- und Implementierungsphasen zusammenfassend dargelegt.

5.1. Vorgehensweise

Die theoretischen Grundlagen zu diesem Abschnitt finden sich in Abschnitt 2.4. Der Softwareentwicklungsprozess und die Vorgehensweise bei der Implementierung werden durch folgende Ziele bestimmt:

- Flexibilität bezüglich zukünftiger Anforderungen während der Projektlaufzeit und nachfolgender Projekte
- ständige Überprüfbarkeit, Lauffähigkeit und Lesbarkeit des Quelltextes
- minimale Komplexität
- Zuverlässigkeit und Robustheit

Die Erreichung dieser Ziele wird unterstützt durch der Implementierung vorangehende visuelle Modellierung mittels *UML* und durch Anwendung einer iterativ inkrementellen Vorgehensweise, welche in Abschnitt 2.4.2 vorgestellt wird. Insbesondere die Lesbarkeit und Verringerung der Komplexität des Quelltextes wird erreicht mit der Durchführung von *Code-Reviews*, teilweisem *Pair-Programming*, automatisierten Tests und vor allem mit der Anwendung von Entwurfsmustern.

Nach der Festlegung auf diese Techniken besteht der zweite Schritt im Erkennen der grundlegenden Struktur (siehe Abschnitt 5.2), welche die Software besitzen muss, um zum einen konform zur *KIEL*-Struktur und *JGraph*-Struktur zu sein und zum anderen den geforderten Funktionsumfang implementieren zu können und dem Softwaretypus entsprechen zu können. Bei der *KIEL*-Struktur sind von wesentlicher Bedeutung die Datenstruktur sowie mit untergeordneter Relevanz die Schnittstellen zum Gesamtsystem *KIEL*. Bei der *JGraph*-Struktur sind die verwendeten Entwurfsmuster, die Datenstruktur sowie die Aufsetzpunkte (engl. *Hooks*) zum Erweitern des Funktionsumfangs interessant.

Nach Analyse der Struktur werden Klassendiagramme entworfen, in denen beschrieben wird, welche Komponenten benötigt werden, wie Funktionalitäten und Verantwortlichkeiten auf Klassen verteilt werden und in welchem Verhältnis die Klassen zueinander stehen. Bei diesem Schritt ist das Erkennen von Problemen notwendig, für die es schon Lösungen gibt, so genannte Entwurfsmuster (Abschnitt 2.4.3). Implementierungsdetails spielen an dieser Stelle noch keine Rolle. Diese werden in Abschnitt 5.3 dargestellt.

5.2. Design

Zunächst werden für die zu erstellende Anwendung grundsätzliche Strukturen festgelegt, welche sich aus den nachfolgenden Kriterien ergeben.

5.2.1. Rahmenbedingungen

Im Folgenden werden die Ergebnisse der grundlegenden Strukturerkennung dargelegt, die sich aus den Rahmenbedingungen ergeben.

Softwaretypus

Bei der zu entwickelnden Software handelt es sich um eine nicht-verteilte Anwendung mit graphischer Benutzeroberfläche, welche in einer nichtkritischen Umgebung eingesetzt wird. Für die Implementierung bedeutet dies, dass das *Model-View-Controller*-Entwurfsmuster Anwendung finden wird.

Die Software ist nicht abhängig von externen (Software-) Systemen, es gibt keine zentralen und dezentralen Komponenten, daher sind Client/Server-Strukturen und andere Kommunikationsstrukturen nicht vorhanden. Des Weiteren werden an die Anwendung weder *erhöhte* Anforderungen bezüglich Robustheit und Sicherheit gestellt – anders als dies bei eingebetteten Systemen z. B. der Fall ist – noch ist das zeitliche Verhalten und der Speicherverbrauch von überdurchschnittlicher Bedeutung.

Struktur des *KIEL*-Projektes

Bei *KIEL* handelt es sich um ein Projekt, welches sich aus mehreren eigenständigen Arbeiten im Rahmen von Studien- und Diplomarbeiten zusammensetzt. Folglich besteht der Gesamtumfang des Softwaresystems *KIEL* zum einen aus voneinander weitgehend unabhängigen Modulen, welche für sich allein lauffähig sind, und zum anderen in geringerem Umfang aus Ressourcen, Funktionalitäten und der *Statechart*-Datenstruktur, welche von diesen Modulen gemeinsam verwendet werden. Folgende Module sind neben dem Editor voneinander unabhängig:

- *Browser* - Anzeige verschiedener *Statechart*-Sichten [50]
- *Layouter* - Layoutalgorithmen auf der *KIEL*-Datenstruktur [28] [29]

- *Simulator* - Simulation von *Statecharts* auf der *KIEL*-Datenstruktur [37]
- *File-Interface* - Interpretation von *Esterel-Studio*-Dateien [51] und Matlab-Dateien [44]

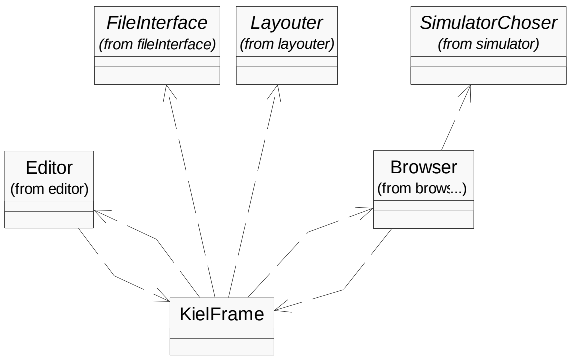


Abbildung 5.1.: Projektstruktur

Unabhängigkeit bedeutet hier – wie in der Abbildung 5.1 zu sehen –, dass zwar der Browser den Simulator nutzt, dass jedoch Editor, Browser, Layouter und das File-Interface nicht gegenseitig voneinander abhängig sind und dass die Schnittstellen zwischen den Modulen sehr schmal sind. Dies hat zur Folge, dass Browser, Editor, Layouter, Simulator und Dateischnittstelle voneinander vollständig entkoppelt werden können, indem auf die jeweilige Funktionalität verzichtet wird. Außerdem sind diese Module auch ohne die Funktionalität der anderen Module sinnvoll verwendbar. Für den Editor bedeutet das z. B., dass ohne Verwendung der Dateischnittstelle die gezeichneten Diagramme ausdruckbar, speicherbar und ladbar bleiben, dass aber auf die Konvertierung von und nach *Esterel-Studio*-Format verzichtet werden muss. Unterbleibt im Weiteren die Verwendung des Layouter-Moduls, so kann der Editor kein automatisiertes Layout durchführen, wohl aber eigene Layoutmechanismen weiterhin einsetzen.

Die voneinander unabhängigen Module Browser und Editor können über die Komponente `KielFrame` miteinander kommunizieren. Diese Komponente stellt das Anwendungsfenster zur Verfügung, bettet die beiden Module in das Fenster ein, übernimmt das Laden von Dateien für diese beiden Module, indem es auf das Modul *File-Interface* zugreift und stellt eine Schnittstelle zur `Layouter`-Komponente zur Verfügung.

Für die Implementierung des Editors bedeutet das, dass dieser nur über eine schmale Schnittstelle Funktionalität anderer Komponenten nutzt und seine Daten nur über das `KielFrame` mit dem Browser austauscht.

Die *KIEL*-Datenstruktur

Die *KIEL*-Datenstruktur lässt sich einteilen in drei Teile. In einem Teil wird die Struktur eines *Statecharts* beschrieben. In einem weiteren Teil werden die Lay-

outinformationen in der Komponente `View` zusammengefasst. Hierbei wird unterschieden zwischen den Layout-Informationen von Knoten, Kanten und ihren Beschriftungen. In einem dritten Teil der *KIEL*-Datenstruktur werden Ereignisse und Bedingungsausdrücke modelliert.

Abbildung 5.2 illustriert den Teil der *KIEL*-Datenstruktur, welcher die Struktur eines *Statecharts* beschreibt. Jedes Diagrammelement ist entweder ein `Node`, eine `Delimiter-Line` oder eine `Edge`. Bei `Edges` z. B. wird mit Hilfe unterschiedlicher Klassen unterschieden zwischen *Strong-Abortions*, *Normal-Terminations* und anderen Transitionstypen. Jedes Element eines *Statecharts* wird durch eine Instanz einer dieser Klassen repräsentiert, z. B. ist ein *Initial-State* im *KIEL*-Datenmodell ein Objekt der Klasse `InitialState`.

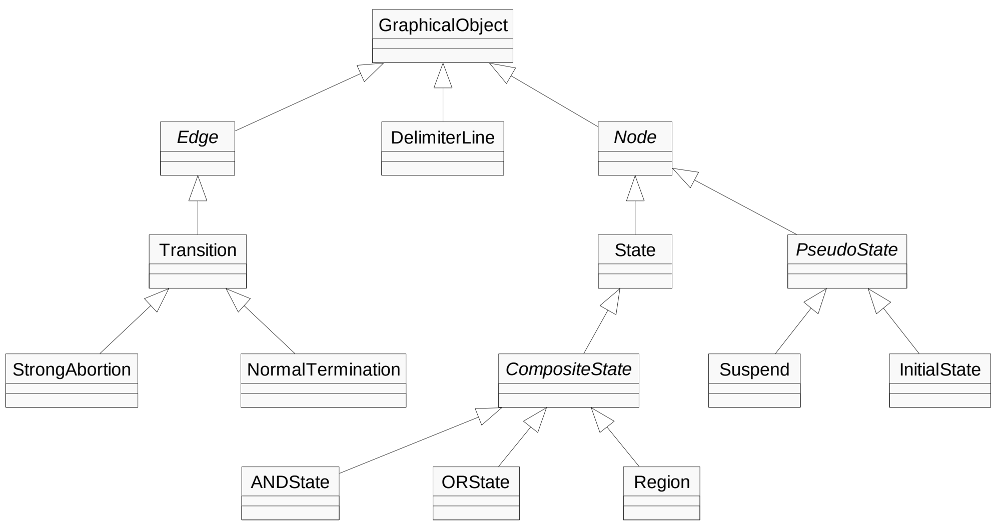


Abbildung 5.2.: *KIEL*-Datenstruktur

Wie in Abschnitt 2.3.2 beschrieben, unterscheiden sich *Composite-States* von allen anderen Diagrammelementtypen dadurch, dass *Composite-States* weitere Komponenten beinhalten können, also durch *Composite-States* eine Baumstruktur aufgebaut werden kann. Daher beinhalten in der *KIEL*-Datenstruktur die `Composite-State`-Komponenten eine Liste von Kind-Knoten. Transitionen besitzen dagegen einen Quellknoten und einen Zielknoten und sind auf diese Weise in die Baumstruktur eines *Statecharts*integriert.

Die graphische Repräsentation der unterschiedlichen Diagrammelementtypen wird durch eine Konfigurationsdatei der *KIEL*-Anwendung spezifiziert und muss von Browser und Editor verwendet werden. Ziel ist eine einheitliche graphische Repräsentation der *Statecharts* durch den Browser und Editor.

Struktur des verwendeten *Graph-Editing-Frameworks*

Die grundlegende Architektur der *JGraph*-Komponente ist die gleiche wie die der Standard-*Swing*-Komponenten: Sie basiert auf dem *MVC*-Muster (Abschnitt 2.4.3). Das *JGraph-Model* beinhaltet dabei neben dem Datenmodell auch auf diesem definierte Operationen. Unter *Datenmodell* werden die das *Statechart* beschreibenden Daten in ihrer *JGraph*-spezifischen Datenstruktur verstanden. Die *JGraph*Datenstruktur ist wie die *KIEL*-Datenstruktur baumförmig aufgebaut. Sie besteht aus `javax.swing.TreeNode`-Komponenten. Jeder Knoten und jede Transition des anzuzeigenden Graphen wird von solch einem `TreeNode`-Objekt repräsentiert, wobei ein `TreeNode`-Objekt genau einen Vater hat und Kinder haben kann. In der *JGraph*-Datenstruktur werden also Transitionen im Gegensatz zur *KIEL*-Datenstruktur als Kinder von Knoten modelliert. Dieser Umstand ist bei der Implementierung der Datenmodell-Transformation zu beachten. Das Design des *JGraph*-Rahmenwerks wird in [1] und [15] beschrieben.

Bei der Nutzung des Rahmenwerks muss für die `JGraph`-Komponente neben einer *View* und einem *Controller* ein *Model* implementiert werden. Dieses *Model* kann aufgrund der Modularisierung eine Repräsentation des *Statecharts* sowohl in Form eines *JGraph*-Datenmodells als auch zusätzlich oder stattdessen in Form eines *KIEL*-Datenmodells beinhalten. Bei einer Hybridlösung ist bei der Implementierung darauf zu achten, dass die notwendigen Transformationen zwischen den beiden Datenmodellen ausschließlich innerhalb des *Models* durchgeführt werden.

5.2.2. Entscheidungsfaktoren

Die Entscheidung für bestimmte Strukturen, Mechanismen und die Verwendung bestimmter Entwurfsmuster ist abhängig von den Rahmenbedingungen des vorhergehenden Abschnitts und von Erfahrungswerten. Das Design der Anwendung ist daneben auch abhängig von einigen an die vorliegende Arbeit gestellten Anforderungen (Kapitel 4). Diese Anforderungen sind vor dem Hintergrund des Anwendungsentwurfs zu bewerten. Die Ergebnisse hierzu werden in diesem Abschnitt dargestellt.

Modellorientierung

Alle Komponenten der *KIEL*-Anwendung arbeiten auf einem gemeinsamen Datenmodell.

Der Einsatz des *MVC*-Entwurfsmusters dient im *KIEL*-Projekt im Besonderen der Kommunikation des Editors mit anderen Projektkomponenten. Der Datenaustausch innerhalb von *KIEL* erfolgt ausschließlich auf der *KIEL*-Datenstruktur, welche auf einer Trennung von *Statechart*-Struktur und seiner Layout-Informationen beruht. Die Kommunikation des Editors mit anderen Projektteilen ist notwendig, um die Dateischnittstelle von *KIEL* nutzen zu können und um die Integration des Editors in das Gesamtprojekt zu ermöglichen.

Da das Rahmenwerk *JGraph* seinerseits eine eigene Datenstruktur besitzt (siehe Abschnitt 5.2.1), muss eine Transformation zwischen beiden Datenmodellen erfolgen. Der Editor wird entweder mit einem von der *KIEL*-Anwendung übergebenen *Statechart* aufgerufen oder es wird ein neues *Statechart* angelegt. Im ersten Fall muss bei Aufruf des Editors eine Transformation vom *KIEL*-Datenmodell in das *JGraph*-Datenmodell erfolgen. Im anderen Fall muss eine Transformation in der umgekehrten Richtung spätestens am Ende des Editiervorgangs durchgeführt werden.

Verschiedene Ansätze zur Umsetzung der Transformation sowie des Transformationszeitpunkts und -umfangs sind hierbei denkbar, wobei zu jedem Zeitpunkt das *KIEL*- und das *JGraph*-Datenmodell nebeneinander existieren:

1. Eine komplette Transformation des Datenmodells wird bei Start des Editors durchgeführt. Es wird hiernach nur noch auf dem *JGraph*-Datenmodell gearbeitet und das *KIEL*-Datenmodell wird nicht mehr verwendet. Aktualisiert wird das *KIEL*-Datenmodell bei Beendigung des Editierprozesses. Nachteilig wirkt sich hierbei jedoch aus, dass auch nicht den Editor betreffende *KIEL*-Datenstruktur-Änderungen Anpassungen am Editor notwendig machen.

2. Es findet keine Transformation sondern lediglich ein *Wrapping* des *KIEL*-Datenmodells durch das *JGraph-Model* statt: Das *Model* besitzt bei dieser Variante keine Repräsentation des *Statecharts*in der *JGraph*-Datenstruktur, sondern alle Operationen des *Models* arbeiten auf dem gegebenen oder initial erzeugten *KIEL*-Datenmodell. Vorteil hierbei ist, dass keine Datenmodell-Transformationen notwendig sind. Allerdings setzt diese Variante einerseits eine genaue Kenntnis der Implementierung des *JGraph-Models*, der Klasse `DefaultGraphModel`, voraus, und andererseits kann der *Undo-and-Redo*-Mechanismus von *JGraph* nicht mehr verwendet werden, da dieser für die *JGraph*-Datenstruktur entwickelt ist.

3. Es wird zu Beginn des Editierprozesses ein *JGraph*-Datenmodell aufgebaut. Sofern ein *KIEL*-Datenmodell vorliegt, findet eine Transformation statt. Dabei enthält das *JGraph*-Datenmodell nur die Editor-relevanten Informationen. Eine vollständige Transformation wird nicht durchgeführt. Beide Datenmodelle sind daher nicht gleichwertig. Das *JGraph-Model* operiert dann auf einem *JGraph*-Datenmodell und einem *KIEL*-Datenmodell. Dabei ändert jede Operation beide Datenmodelle. Der Vorzug dieser Variante ist, dass die Implementierung unabhängig von Editor-irrelevanten *KIEL*-Datenstruktur-Änderungen ist.

4. Es werden nur die hierarchische Struktur des *Statecharts* sowie die Diagrammelementtypen und graphischen Informationen transformiert. Daten, welche in *JGraph* nicht vorgesehen sind, z. B. das `isCollapsed`-Attribut von *Composite-States*, werden weiterhin im *KIEL*-Datenmodell gehalten. Vorteilhaft erscheint hierbei, dass einige Daten nicht doppelt abgelegt werden und damit nicht synchronisiert werden müssen.

Es wurde die vierte Variante gewählt. Hierbei müssen nur diejenigen Daten aus dem *KIEL*-Datenmodell gelesen und betrachtet werden, welche für den Editor relevant sind. Bei Änderung der Struktur der für den Editor irrelevanten Daten ist eine Änderung der Implementierung des Editors nicht notwendig. Des Weiteren spiegelt dieser Ansatz die Tatsache wider, dass beide Datenstrukturen sich nur in der Repräsentation der hierarchischen Struktur des *Statecharts*ähneln, (siehe sec:jgraphStruktur). Ob ein Zustand momentan kollabiert ist (`isCollapsed`) und welche Elemente wie *Events* und Variablen dieser besitzt, kann nicht sinnvoll in der *JGraph*-Datenstruktur modelliert werden. Insgesamt betrachtet stellt die vierte Variante auch den intuitiven Lösungsansatz dar. Das *JGraph*-Rahmenwerk wird hierbei auf eine Art und Weise erweitert, welche in der *JGraph*-Dokumentation vorgeschlagen wird, siehe 5.7. Allerdings ist bei diesen Überlegungen zu beachten, dass der *Undo*-Mechanismus des Rahmenwerks verwendet werden soll, welcher ausschließlich auf dem *JGraph*-Datenmodell operiert. Soll z. B. das Kollabieren eines *Composite-States* mittels *Undo*-Funktionalität rückgängig gemacht werden können, so muss das `isCollapsed`-*Flag* in die *JGraph*-Datenstruktur integriert werden, um Änderungen dieses *Flags* registrierbar für den *Undo*-Mechanismus zu gestalten. Das gleiche gilt für *Events* und Variablen eines *Statecharts*. Es wird entschieden, dass zwar `isCollapsed`-Änderungen durch den *Undo*-Mechanismus rückgängig gemacht werden können, nicht jedoch soll dies für das Editieren von *Events* und Variablen gelten.

Graphische Information im *SVG*-Format

Scalable-Vector-Graphics (SVG) ist eine in der *Extensible-Markup-Language (XML)* formulierte Sprache zur Beschreibung zweidimensionaler Vektorgraphiken. Der *KIEL*-Browser verwendet ein *SVG*-Rahmenwerk zur Darstellung von *SVG*-Objekten, welche ein *Statechart* repräsentieren. Ein *SVG*-basierter Ansatz ist im Editor wünschenswert, um die graphische Repräsentation von *Statecharts* auf die gleiche technische Weise zu realisieren und damit eine zwischen Browser und Editor exakt gleiche Darstellung von *Statecharts* zu erreichen, vergleiche hierzu Abschnitt 5.2.1.

Bei der Frage, ob im *KIEL*-Editor für die Darstellung graphischer Informationen das *SVG*-Format gewählt werden kann, muss unterschieden werden zwischen zwei Ansätzen: Zum einen ist vorstellbar, ein *SVG*-Rahmenwerk einzusetzen, welches als Grundlage für die Darstellung *und* die benutzergesteuerte Bearbeitung von *Statecharts* verwendet wird. Alternativ ist denkbar, ein *SVG*-Rahmenwerk zusammen mit *JGraph* zu verwenden, wobei das *SVG*-Rahmenwerk die Darstellung des Diagramms übernimmt und *JGraph* die Benutzerinteraktionen.

Gegen die erste Variante spricht,dass keine *SVG*-Rahmenwerke verfügbar sind, welche die Bearbeitung von *SVG*-Objekten in Form von grundlegenden benutzergesteuerten Graph-Bearbeitungsfunktionen (siehe Abschnitt 3.2.1) erlauben, welche ihrerseits in Diagramm-Editoren üblicherweise auftreten. Gegen die zweite Variante spricht, dass die verfügbaren *SVG*-Rahmenwerke und das für den Editor zur Verwendung vorgesehene Rahmenwerk nicht miteinander vereinbar sind. SVG stellt

eine Abstraktionsschicht über der des *Java-Abstract-Windowing-Toolkit* (AWT) dar. *JGraph* verwendet direkt dieses *AWT*. Eine Zwischenschicht wie die des *SVG* lässt sich daher zusammen mit *JGraph* nicht nutzen.

5.2.3. Ergebnisse

Dieser Abschnitt zieht Schlüsse aus den vorhergehenden Abschnitten über die Rahmenbedingungen und Entscheidungsfaktoren und konkretisiert das Design für die einzelnen Komponenten des Editors. Hierbei handelt es sich um Ergänzungen zu den vorhergenden Abschnitten.

Der Editor basiert in seiner grundlegenden Struktur auf dem *MVC*-Muster, wird also aus den Komponenten *Model*, *View* und *Controller* bestehen. Der *Controller* wiederum wird nach dem Zustandsmuster und dem Befehlsmuster aufgebaut sein. Abbildung 5.3 zeigt die drei *MVC*-Komponenten *Model*, *Controller* und *View* in einer Anordnung von links nach rechts. Das *Model* besteht aus den Klassen `NodePropertiesDialog`, `UndoClusterManager` und `MyGraphModel`. Die *View* wird gebildet aus den Klassen `KielFrame`, `Editor`, `MyJGraph` und weiteren in der Abbildung nicht gezeigten Komponenten. Die `ActionXY`-Klassen modellieren den *Controller* des *KIEL*-Editors. Die Pfeile in der Abbildung 5.3 zeigen die Abhängigkeiten der Klassen untereinander. Da jede der in dieser Abbildung gezeigten Klassen nur ein Mal instantiiert wird, formuliert dieses Klassendiagramm gleichzeitig die Objektbeziehungen und damit die Kommunikationswege. Wie in Abschnitt 2.4.3 beschrieben sind hier keine bidirektionalen Kommunikationskanäle zu finden. Benutzerereignisse werden von der *View* an die jeweiligen *Controller*-Komponenten weitergeleitet, welche ihrerseits hauptsächlich auf dem `MyGraphModel` operieren. Eine Änderung dieses *Models* verursacht eine Aktualisierung des `Editor`-Objekts und des `MyJGraph`-Objekts. Diese Klassen werden in den folgenden Abschnitten weiter erläutert.

Das *Model*

Das *Model*, implementiert durch die Klasse `MyGraphModel`, kapselt die Transformation zwischen den Datenmodellen von *KIEL* und *JGraph* und beinhaltet zu jedem Zeitpunkt das komplette *KIEL-Model* und das *JGraph-Model* des *Statecharts*. Bei Start des Editors mit einem im *KIEL*-Datenmodell vorhandenen *Statechart* wird eine Transformation in das *JGraph*-Datenmodell durchgeführt.

Für jedes *KIEL-Statechart*-Diagrammelement-Objekt existiert ein korrespondierendes *JGraph*-Diagrammelement-Objekt. Beide Elemente sind miteinander verbunden: Einerseits besitzt jedes der *JGraph*-Diagrammelement-Objekte eine Referenz auf das zugehörige *KIEL*-Objekt, andererseits wird ein *Mapping* aufgebaut, welches zu einem *KIEL*-Objekt das zugehörige *JGraph*-Objekt liefert. Diese Verknüpfungen sind notwendig, weil bei einigen Operationen auf dem *JGraph*-Datenmodell auch Daten aus dem *KIEL*-Datenmodell gebraucht werden. Daneben werden auch Operationen implementiert, welche auf dem *KIEL*-Datenmodell

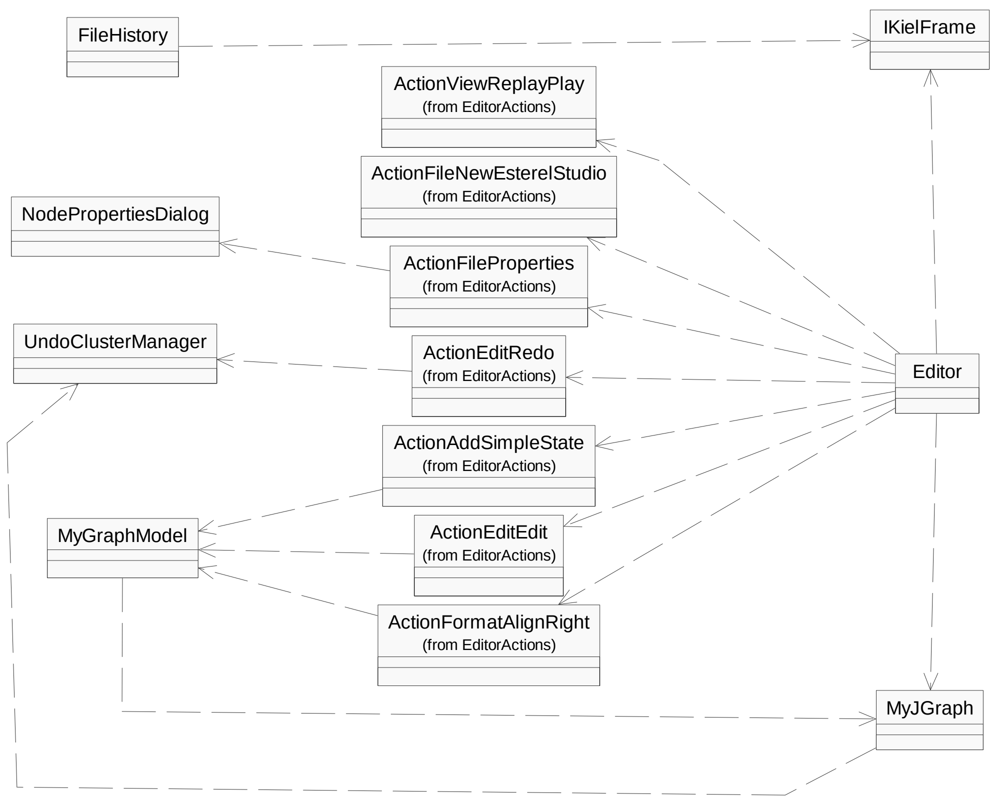


Abbildung 5.3.: Das *Model-View-Controller*-Entwurfsmuster

arbeiten und hierbei auf Daten der entsprechenden *JGraph*-Diagrammelemente zugreifen müssen. Ein Beispiel für solche Operationen wird in Abschnitt 5.3.2 beschrieben.

Änderungen an den *JGraph*-Elementen können auf zwei verschiedene Arten erfolgen:

1. Das *JGraph*-Element wird durch das Rahmenwerk direkt geändert. Hierbei findet keine Aktualisierung des zugehörigen *KIEL*-Elementes statt. Solche Änderungen führen zu temporären Inkonsistenzen zwischen beiden *Models*, welche bei Bedarf explizit aufgelöst werden müssen. Dieser Bedarf besteht z. B. bei Ausführung eines Diagrammelementtyp-Morphings (Abschnitt 5.3.3), da hier anhand der gültigen Daten eines *KIEL*-Objekts ein neues *KIEL*-Objekt erzeugt werden muss.

2. Das *Statechart* wird über das *Model* (Abschnitt 2.4.3) geändert. Bei jeder solcher Änderungen wird die Änderung auf beiden Datenmodellen durchgeführt und es entstehen keine Inkonsistenzen.

Die Synchronisation auf Anfrage reicht aus, um ein ordnungsgemäßes Arbeiten mit der Anwendung zu ermöglichen. Nähere Ausführungen hierzu sind im Anhang im Abschnitt A.4 im Programm-Quelltext zu finden.

Der *Controller*

Der *Controller* definiert die Reaktionen auf Benutzereingaben und besteht aus Befehlsklassen, welche die Namen `ActionXXX` und `EditorModeXXX` tragen, siehe Abbildung 5.3 und 5.4. Zur Entscheidung, welches Entwurfsmuster innerhalb des *Controllers* angewendet werden soll, tragen aus Implementationssicht folgende Anforderungen und Faktoren bei:

- Benutzeraktionen, welche Layout-Aufträge beinhalten, werden eine längere Zeit in Anspruch nehmen, daher müssen Benutzeraktionen grundsätzlich in einer Schlange abgearbeitet werden können.
- Es sollen Benutzereingaben protokolliert werden.
- *Undo-and-Redo*-Funktionaliät soll implementiert werden.
- Die Menge an ausführbaren Befehlen soll leicht änderbar sein während der Entwicklungsphase.
- Einige Benutzeraktionen bestehen aus mehreren Bearbeitungs-Teilschritten. Jeder dieser Teilschritte wird also im Kontext eventuell vorhergehender Teilschritte durchgeführt.

Das Befehl-Entwurfsmuster eignet sich für die ersten vier genannten Punkte, da bei diesem Muster alle Benutzeraktionen als Befehlsobjekte implementiert werden und als solche gut gespeichert und sukzessive abgearbeitet werden können. Der *Controller* besteht daher aus einer Vielzahl solcher Befehlsklassen, welche eine einheitliche Aufrufschnittstelle besitzen werden und eine Referenz zum *Model* besitzen.

Des Weiteren wird es auch Benutzeraktionen geben, welche nicht aus einem einfachen Mausklick, sondern aus einer Gruppe hintereinander folgender Teilschritte bestehen. Wenn der Benutzer z. B. ein Diagrammelement hinzufügen möchte, dann besteht die Aktion darin, den Mauszeiger über der Zeichenfläche zu bewegen und dabei den Editor permanent prüfen zu lassen, ob die aktuelle Mausposition erlaubt ist, und dann eine bestimmte Position zu bestimmen und das Element dort zu platzieren. Benutzer eines graphischen Editors sind daran gewöhnt, für eine solche Aktion in einen Einfüge-Modus zu wechseln. Es wird dabei auf einen Button in einer Werkzeugleiste geklickt und damit ändert sich der Mauszeiger und zeigt an, in welchem Modus sich der Editor nun befindet, welche Bedeutung also die nächste Mausbewegung hat. Im Editor wird es Aktionen geben, welche ihn in einen bestimmten Zustand versetzen, in dem der Editor dem Zustand entsprechend reagiert.

Hierfür findet das Zustands-Entwurfsmuster Anwendung. Beachtet werden muss hierbei allerdings, dass jeder auf diese Weise modellierte Editor-Zustand gleichzeitig eine Benutzeraktion ist. Daher wird für jeden Editor-Zustand eine Klasse modelliert, welche ihrerseits eine Befehlsklasse (`EditorAction`) ist, siehe hierzu Abbildung 5.4.

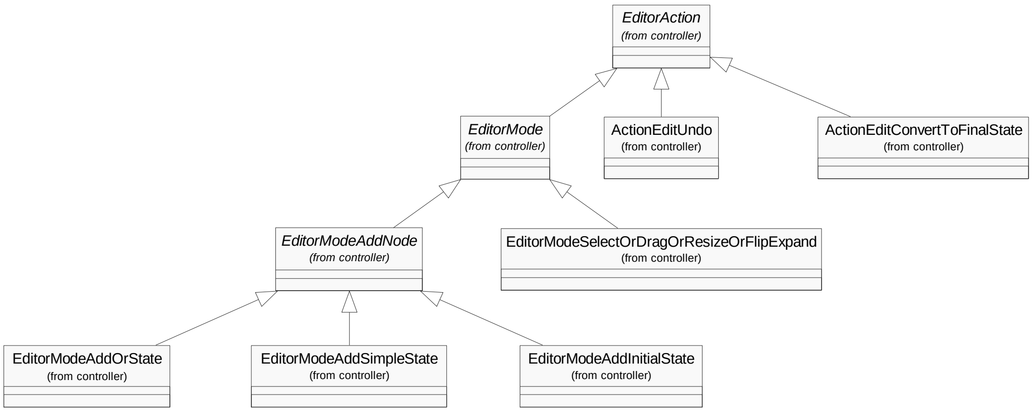


Abbildung 5.4.: Der *Controller* des *MVC*-Musters

Die *View*

Die *View* beinhaltet die graphische Benutzeroberfläche des Editors. Hierzu zählen die Zeichenfläche, die Werkzeugleiste und Teile der Menü-Leiste, nicht aber das *KIEL*-Anwendungsfenster, welches eine eigenständige Komponente ist, die vom Editor-Modul getrennt ist, siehe Abschnitt 4.20.

Die Zeichenfläche des Editors ist eine Benutzeroberflächen-Komponente, welche das *JGraph*-Rahmenwerk bereitstellt. Zu den Werkzeugleisten-Buttons und den editorspezifischen Menü-Einträgen gehören eigenständige *Controller*, siehe 5.3 und 5.4. Für die *View*-Komponente müssen keine grundlegenden Design-Entscheidungen getroffen werden.

Das *KIEL*-Anwendungsfenster

Das Hauptfenster von *KIEL* soll eine eigenständige Komponente sein, welche das Editor-Modul und das Browser-Modul zusammenführt und miteinander verbindet, so dass das angezeigte *Statechart* zwischen diesen Komponenten ausgetauscht werden kann, jedoch ohne dass Editor und Browser voneinander anhängig sind. Die Entkopplung von Komponenten wird mit Hilfe der Definition von Schnittstellen erreicht: einer Schnittstelle (`KielComponent`) für die Beschreibung von Editor und Browser sowie einer Schnittstelle (`IKielFrame`, siehe Abbildung 5.5) für die Beschreibung des *KIEL*-Hauptfensters. Dieses Hauptfenster soll Funktionalitäten bereitstellen, welche von Browser und Editor gleichsam verwendet werden, siehe

Abschnitt 4.20. Diese Funktionalitäten prägen die Schnittstelle `IKielFrame`, siehe auch Abbildung 5.5.

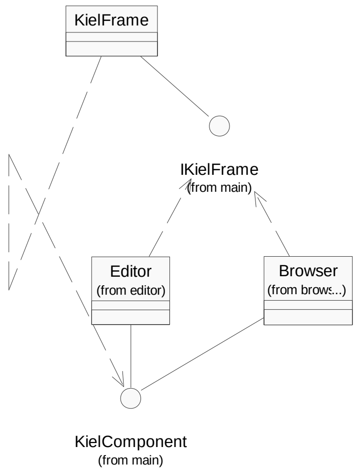


Abbildung 5.5.: Entkopplung der *KIEL*-Komponenten

Konfigurierbarkeit

In vielen Bereichen des Softwareentwurfs müssen Annahmen getroffen werden über die Bedürfnisse und Anwendungsgewohnheiten der Benutzer. Beispielsweise muss festgelegt werden, in welcher Sprache die Benutzeroberfläche mit dem Benutzer kommunizieren soll und es muß geschätzt werden, welchen Zoomfaktor bei der Anzeige von Objekten der Benutzer erwartet. Anstatt diese Dinge zur Entwicklungszeit bereits festzulegen und damit die Anwendung nur für einen Teil der Benutzer optimal zu gestalten, ist es vorteilhafter, solche Eigenschaften der Software variabel zu halten und dem Benutzer die Möglichkeit zu geben, sich die Anwendung nach den eigenen Bedürfnissen zu konfigurieren, siehe Abschnitt 4.22.

Folgende Designentscheidungen müssen für den Editor getroffen werden:

- Welche Eigenschaften sollen konfigurierbar sein?
  Es können die in Abschnitt 4.22 aufgezählten Eigenschaften konfiguriert werden. Dies hat zur Folge, dass an entsprechenden Stellen im Programm auf variable Werte zugegriffen wird.

- Wann sollen sie konfigurierbar sein?
  Der Benutzer soll zu jedem Zeitpunkt während des Programmlaufs die Anwendungseigenschaften einstellen können, nicht jedoch nach Beendigung des Programms. Die Änderungen sollen darüber hinaus nach Beendigung des Programms erhalten bleiben. Notwendig ist also eine Konfigurationsdatei.

- Wann sollen Konfigurationsänderungen aktiv werden und wie sollen diese durchgeführt werden?
  In anderen Anwendungen können Konfigurationsdateien per Hand geändert werden, was für den Durchschnittsbenutzer unkomfortabel ist und ein Aktivwerden vieler Eigenschaften erst nach erneutem Start der Anwendung zulässt, da die Anwendung Änderungen an diesen Dateien nicht registriert. Im Editor sollen hingegen solche Änderungen sofort angewendet werden. Dies erfordert ein erhöhtes Maß an Flexibilität und Komplexität der Anwendung. Zunächst ist hierfür erforderlich, dass Änderungen nur über die Benutzeroberfläche erfolgen dürfen, damit Benutzeränderungen vom Programm registriert werden können. Des Weiteren müssen alle Softwarekomponenten benachrichtigt werden über benutzergesteuerte Konfigurationsänderungen. Hierfür eignet sich das Beobachter-Entwurfsmuster, siehe Abschnitt 2.4.3. Um z. B. die Änderung der Sprache der Benutzeroberfläche zu veranlassen, muss jedes Benutzeroberflächen-Objekt, welches Text anzeigt – dies sind Buttons, *Tooltips*, Menü-Einträge und andere – eine Beobachter-Schnittstelle implementieren und sich als Beobachter anmelden. Gleiches gilt für Benutzeroberflächen-Komponenten, deren Farbe einstellbar sein soll und für die Zeichenfläche, für die Eigenschaften wie Rastergröße und Handle-Größe konfiguriert werden sollen.

- Wie soll mit ungültigen Konfigurationswerten umgegangen werden?
  Für jeden konfigurierbaren Wert muss ein Standardwert definiert sein. Dieser wird immer dann herangezogen, wenn der Benutzer entweder den Wert noch nicht spezifiziert hat oder wenn er einen ungültigen Wert eingegeben hat. Ein Wert soll der Einfachheit halber schon dann gültig sein, wenn sein Datentyp passt. Bereichsprüfungen werden nicht durchgeführt.

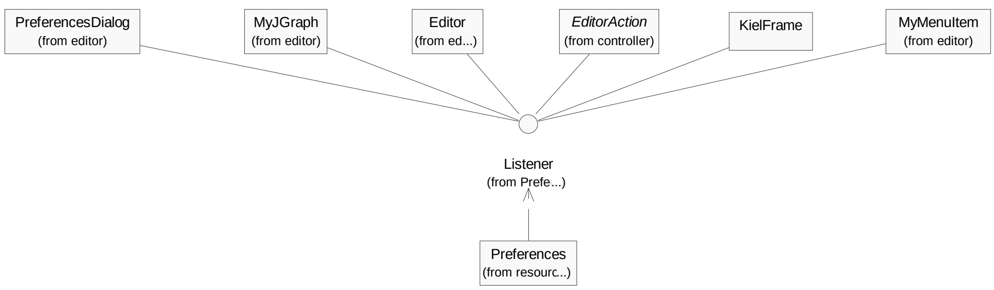


Abbildung 5.6.: Benachrichtigungsmechanismus bei Konfigurationsänderungen

Abbildung 5.6 zeigt die Klasse `Preferences`, in welcher Konfigurationseinstellungen verwaltet werden. Bei jeder Änderung sendet sie allen `Listener`-Objekten eine Nachricht hierüber, woraufhin die benachrichtigten Objekte die für sie relevanten Einstellungen abrufen. Zu den Interessenten von Konfigurationseinstellungsänderungen gehören:

- das Dialogfenster zur Eingabe der Einstellungen (`PreferencesDialog`)
- die *Statechart*-Darstellungsfläche (`MyJGraph`)
- die Editor-*View*-Wurzelklasse (`Editor`) mit ihrer ausblendbaren Übersichtsschicht
- alle *Controller*-Klassen (`EditorAction`), deren im *Tracelog* dargestellter Name sprachabhängig ist (siehe Abschnitt 4.9)
- alle Menüeinträge (`MyMenuItem`), die ebenfalls sprachabhängig sind

5.3. Implementierung

In diesem Abschnitt wird auf die Umsetzung des im vorherigen Abschnitt entworfenen Designs eingegangen. Dabei werden einige Implementierungsdetails interessanter Funktionalitäten aufgezeigt. Zugunsten der Übersichtlichkeit und Verständlichkeit wird darauf verzichtet, auf sämtliche Implementierungsdetails und technische Probleme einzugehen. Weitergehende Details zur Implementierung, auch des gesamten *KIEL*-Projektes, gibt die *Javadoc*-Dokumentation des Projektes [45]. Der Programm-Quelltext wird unter strikter Einhaltung der *Sun-Java-Programing-Guidelines* [43] geschrieben. Dies wird durch das Werkzeug *Check Style* überprüft. Des Weiteren wird *JLint* eingesetzt. Dieses Werkzeug erkennt bei seiner statischen Quelltextanalyse mögliche Fehler und Probleme in der Klassenhierarchie, im Datenfluss und bei der Synchronisation von *Threads*. Sowohl *Check-Style* als auch *JLint* und *Javadoc* erkennen im Programm-Quelltext des Editormoduls nach Beendigung der Implementierungsphase keine Fehler mehr und liefern bei ihrer Analyse auch keine Warnungen mehr.

5.3.1. Erweiterung des *JGraph*-Rahmenwerks

JGraph basiert auf *Java-Swing* und dem *MVC*-Entwurfsmuster. Die nachfolgenden Beschreibungen illustriert Abbildung 5.7. Die *View*-Komponente von *JGraph* besteht unter anderem aus der Klasse `JGraph`, welche in Form einer *Swing*-Komponente einen Graphen darstellt. Das *Model* wird repräsentiert von der Klasse `GraphModel`. Diese verwaltet den Graphen ähnlich wie die `javax.swing.JTree`-Klasse in Form einer Baumstruktur. Diese Baumstruktur ist sowohl in der Klasse `JGraph` als auch in der Klasse `GraphModel` in Form einer Menge von `javax.swing.TreeNode`-Objekten aufgebaut. Jedes konkrete Diagrammelement wird als solch ein `TreeNode`-Objekt dargestellt. Dieses Element ist gleichzeitig entweder vom Typ `DefaultGraphCell`, sofern es das Diagrammelement im *Model* repräsentiert oder vom Typ `EdgeView` oder `VertexView`, sofern es das Diagrammelement in der *View* repräsentiert. Jede *View* stellt eine individuelle Sicht auf den Graphen dar. Das hat zur Folge, dass ein einzelnes Diagrammelement durch ein Objekt im *Model* dargestellt wird und in jeder *View* durch je *View* ein ebenso individuelles Objekt. Zu

einer `EdgeView`- und `VertexView`-Klasse gehört eine Klasse, welche die graphische Repräsentation beschreibt, ein so genannter `Renderer`. In diesem werden die visuellen Eigenschaften des Diagrammelements in Form von Graphik-Primitiven aus `java.awt.Graphics` beschrieben. Weiterführende Informationen zu *Swing* und Graphik-Primitiven bietet [53].

Die unterschiedlichen Diagrammelementtypen von *KIEL* bestimmen die Menge an Klassen, welche in *KIEL* das *JGraph*-Framework erweitern. Alle Klassen in der Abbildung 5.7, welche den Zusatz *from graph* beinhalten, stellen die zu implementierende Erweiterung des *JGraph*-Rahmenwerks dar. Die Klassen `VertexView` und `EdgeView` werden im Design des *KIEL*-Editors die Basisklassen für die Pendents der *KIEL*-Datenstruktur-Klassen `Node` und `Edge` sein. Diese Pendents heißen in der *JGraph*-Erweiterung `NodeView` und `EdgeView`. Diese `XYView`-Klassen repräsentieren die Diagrammelementtypen in der *View*-Komponente von *JGraph*. Die *View* besteht des Weiteren aus Instanzen der Klassen `VertexRenderer` und `EdgeRenderer`. Für diese beiden Klassen müssen ebenso je Diagrammelementtyp Subklassen entworfen werden.

Das *JGraph-Model* hingegen besteht ausschließlich aus Objekten der Klasse `DefaultGraphCell`. Diese Klasse wird die Basisklasse der *KIEL-Datenstruktur*-Klasse `GraphicalObject` sein. Für jede Datenstruktur-Klasse namens `XY` wird eine `XYCell`-Klasse für das *Model* und eine `XYView`-Klasse sowie eine `XYRenderer`-Klasse für die *View* entworfen, wobei die Klassenhierarchie von *KIEL* übernommen wird.

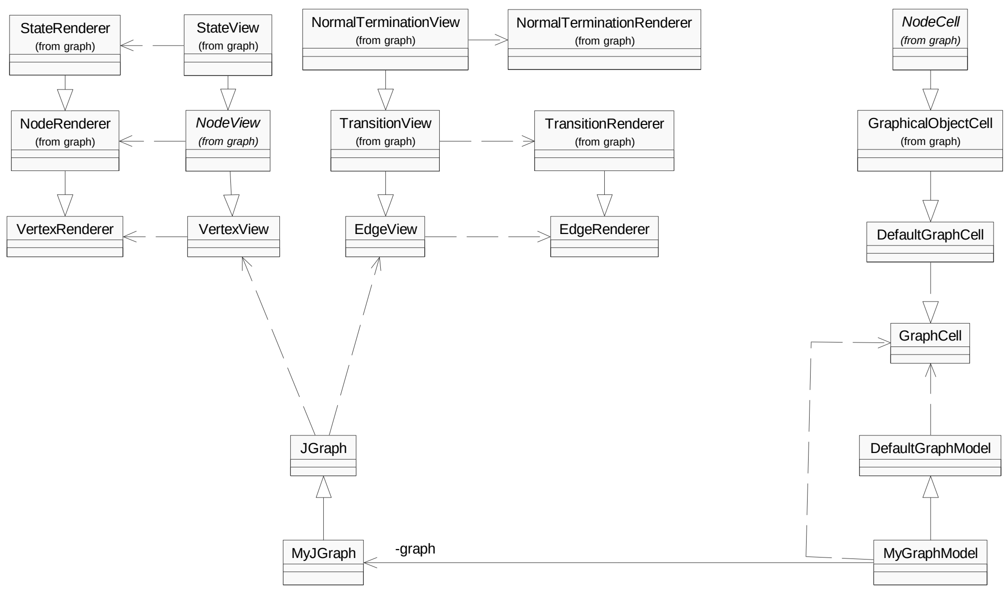


Abbildung 5.7.: Erweiterung des *JGraph*-Rahmenwerks

Für die Anpassung der *JGraph-View* auf *KIEL* muss diese Klasse erweitert wer-

den nach Klasse `MyJGraph`. Die hierfür notwendigen Schritte werden in den nächsten Abschnitten erläutert.

Darstellung verschiedenartiger Diagrammelemente

In der *KIEL*-Datenstruktur wird jede Diagrammelement-Art durch eine eigene Klasse repräsentiert und damit jedes Diagrammelement durch ein individuelles Objekt des entsprechenden Typs dargestellt. Dieser natürliche Ansatz wird auf die umzusetzende Erweiterung des *JGraph*-Rahmenwerks übertragen. Beispielsweise wird für die Klasse `ORState` aus der *KIEL*-Datenstruktur in der *JGraph*-Erweiterung eine `ORStateCell`-Klasse für das *Model* sowie eine `ORStateView`-Klasse für die *View* und eine `ORStateRenderer`-Klasse für die graphische Repräsentation definiert. Die Subklassen von `GraphCell` und `CellView` (siehe Abbildung 5.7) stellen lediglich Typen dar, welche keine Implementierungsdetails beinhalten. Besitzen zwei oder mehrere Diagrammelemente das gleiche Aussehen, so braucht für diese Diagrammelemente nur ein einziger `CellRenderer` implementiert zu werden.

Durch diese Art der Erweiterung des *JGraph*-Rahmenwerks wird die Form der *KIEL*-Datenstruktur auf das *JGraph*-Rahmenwerk übertragen und schafft damit eine geeignete Struktur für die Transformationsaufgaben, wie sie in Abschnitt 5.2.2 beschrieben werden.

Syntax-Überprüfung

Das Ergebnis der Syntax-Überprüfung eines *Statecharts* bestimmt die Rahmenfarbe der einzelnen Diagrammelemente. Die Syntax-Prüfung muss daher immer dann erfolgen, wenn das *Statechart* durch das Rahmenwerk neu gezeichnet wird, wenn also die `CellViewRenderer` aufgerufen werden. Bei diesem Aufruf wird dem `CellViewRenderer` das zu zeichnende `CellView`-Objekt übergeben, welches gezeichnet werden soll. Dieses `CellView`-Objekt besitzt eine Referenz auf die *View*, in welcher es enthalten ist. Die *View* wird implementiert durch die Klasse `MyJGraph`. Nur wenn es sich bei der *View* um diejenige handelt, welche editiert wird, ist der Rahmen des `CellView`-Objektes entsprechend dem Syntax-Überprüfungsergebnis zu zeichnen.

Diese Anforderungen, verbunden mit dem beschriebenen Mechanismus, mit dem das Rahmenwerk den Graphen zeichnet, führen zu der Möglichkeit, die gesamte Syntax-Überprüfung in der `MyJGraph`-Klasse zu implementieren. Grundsätzlich ist bei der Implementierung immer zu prüfen, ob durch die Wahl des Ortes einer Implementierung die Komplexität eines Softwaresystems gering gehalten werden soll und damit Verantwortlichkeiten im Sinne der objektorientierten Programmierung wie in diesem Fall zusammengefasst werden, oder ob der Gedanke der Erweiterbarkeit verfolgt werden soll, was in diesem Falle eine Aufspaltung in Graph-Anzeige und Syntaxüberprüfung bedeuten würde. Die Entscheidung fällt für die erstere Variante. Die Regeln sind in Abschnitt 4.11 angegeben.

Verwaltung der Editor-Modi

Der Editor kennt die Zustände Selektionsmodus sowie die Hinzufüge-Operationen *Transition*, *Deep-History-State*, *Horizontal-Delimiter*, *Vertical-Delimiter*, *Dynamic-Choice*, *Final-Simple-State*, *History-State*, *Initial-State*, *OR-State*, *Simple-State* und *Suspend*.

Wie in Abschnitt 5.2.3 erwähnt, wird zur Umsetzung der Editor-Modi das Zustands-Entwurfsmuster, siehe Abschnitt 2.4.3, verwendet. Es wird eine abstrakte Klasse `EditorMode` entworfen, deren Schnittstelle Maus-Ereignisse entgegen nimmt sowie abfragen lässt, ob der Modus im Moment aktiv (engl. *enabled*) ist. Dies ist bei allen Modi, in denen Zustände hinzugefügt werden, abhängig von der aktuellen Mausposition. Die Klasse `EditorModes` verwaltet die `EditorMode`-Objekte. Zu jedem Zeitpunkt erhält man mit `EditorModes.getCurrent()` das `EditorMode`-Objekt abhängig vom benutzergewählten Zustand. Gesetzt wird das entsprechende `EditorMode`-Objekt, wenn der Benutzer entweder im Menü eine Hinzufüge-Operation auswählt oder im Standard-Modus das Ziehen einer Transition beginnt.

Das `EditorModes`-Objekt muß immer dann reagieren, wenn über der Zeichenfläche – dem von `JGraph` abgeleiteten `MyJGraph`-Objekt – der Mauszeiger bewegt wird. Daher wird die Anbindung des `EditorModes`-Moduls wie folgt realisiert: Am `MyJGraph`-Objekt wird ein `java.awt.event.MouseListener`-Objekt registriert, welches rahmenwerkspezifisch in diesem Fall ein Objekt der Klasse `MyBasicGraphUI.MyMouseHandler` ist. Dieses delegiert in seinen `MouseListener`-Methoden alle Anfragen nach dem *Delegator*-Muster an das `EditorModes`-Objekt, welches über `EditorModes.getCurrent()` erreichbar ist. Welches `EditorMode`-Objekt die Methode `EditorModes.getCurrent()` zurückliefert, hängt davon ab, welchen Modus der Benutzer vorher ausgewählt hat. Der Controller des *Simple-State-Hinzufügen*-Buttons, genannt `EditorModeAddSimpleState`, ruft bei Klick auf den Button die Operation `EditorModes.setCurrent(this)` auf, setzt also sich selbst als aktuellen `EditorMode`.

5.3.2. Datenmodell-Transformation zwischen *KIEL* und *JGraph*

Die Implementierung der Transformation von *Statecharts* aus dem *KIEL*-Format ins *JGraph*-Format und umgekehrt richtet sich nach den Ergebnissen aus den Abschnitten 5.2.2 und 5.2.3. Im nachfolgenden bezeichnet der Begriff *KIEL-Statechart* ein *Statechart* im *KIEL*-Format und analog *JGraph-Statechart* ein *Statechart* im *JGraph*-Format.

Die Transformation findet im *JGraph-Model* statt. Es wird hierzu eine Klasse `MyGraphModel` entworfen. Diese besitzt eine Referenz zum *KIEL-Statechart* und zur *KIEL-View*, welche ihrerseits die Layout-Informationen für das *KIEL-Statechart* beinhaltet. Des Weiteren beinhaltet das `MyGraphModel` eine Referenz zum *Root-State* des *JGraph-Statecharts*, dessen sukzessiver Aufbau im Folgenden behandelt wird.

Das `MyGraphModel` besitzt Operationen zum Hinzufügen von Diagrammelemen-

ten, unter anderem `addDelimiter()`, `addNode()` und `addTransition()`. Zunächst wird der Fall betrachtet, in dem der Editor mit einem *KIEL-Statechart* aufgerufen wird. Der Baum dieses *KIEL-Statecharts* wird traversiert, wobei für jedes *Statechart*-Element ein `GraphCell`-Objekt erzeugt wird. Für die Transformation elementar sind die Methoden `MyGraphModel.add(Node)` und `MyGraphModel.addRegionOrORState(CompositeState)`. Gestartet wird die Transformation mit der Methode `MyGraphModel.add(Node)`, der initial der *Root-State* übergeben wird und welche rekursiv aufgerufen wird. In dieser Methode wird für das übergebene *KIEL*-Objekt ein *JGraph*-Objekt erzeugt und dann in Abhängigkeit davon, ob es sich um einen *OR-State* handelt, die `MyGraphModel.addRegionOrORState()`-Methode aufgerufen, oder, falls es sich um einen *AND-State* handelt, die *Delimiter-Lines* hinzugefügt und für jede *Region* wiederum die `MyGraphModel.addRegionOrORState()`-Methode aufgerufen. Die `MyGraphModel.addRegionOrORState()`-Methode fügt zunächst alle Kinder der *Region* oder des *OR-States* mit der `MyGraphModel.add(Node)`-Methode hinzu, setzt die Vater-Kind-Beziehung und erzeugt danach für jede ausgehende Transition des *Composite-States* ein *JGraph*-Objekt und fügt es als Kind dem *JGraph*-Objekt hinzu. Hier liegt der einzige Unterschied zwischen *KIEL*-Struktur und *JGraph*-Struktur: Die Transitionen hängen in der *KIEL*-Struktur an ihrem Quell- und Zielzustand. In der *JGraph*-Struktur hingegen sind Transitionen *Kinder* eines Zustands. Bei der hier beschriebenen Transformation wird festgelegt, dass im *JGraph*-Modell eine Transition das Kind des Vaters des Quellzustands ist.

Beachtung findet ferner die Tatsache, dass die Koordinaten im *JGraph*-Modell relativ zum Nullpunkt des Koordinatensystems angegeben werden. Im *KIEL*-Modell hingegen werden diese Koordinaten relativ zur linken oberen Ecke des Vater-Zustands beschrieben und die Transitionen werden relativ zur linken oberen Ecke des Quellzustands angegeben. Bei Berechnung der JGraph-Koordinaten wird daher eine Umrechnung erfolgen müssen. Diese Umrechnung wird bei dem betreffenden Diagrammelement starten und rekursiv die Hierarchie in Richtung des *Root-States* traversieren. Dabei werden die relativen Koordinaten der auf dem Weg liegenden Zustände aufaddiert und so die absoluten Koordinaten des betreffenden Zustands erhalten.

5.3.3. Datenmodell-Änderungen

Sofern nachfolgend nicht anders vermerkt, müssen sämtliche Änderungen sowohl auf dem *KIEL*-Datenmodell als auch auf dem *JGraph*-Datenmodell durchgeführt werden. Bei der Implementierung wird darauf geachtet, die Gemeinsamkeiten der Datenmodell-Transformation (Abschnitt 5.3.2) und der Datenmodell-Änderungen herauszuarbeiten, um die Datenmodell-Änderungen auf den Operationen der Datenmodell-Transformation basieren lassen zu können.

Typ-Morphing

Diese Funktionalität erlaubt es, den Typ eines Zustands oder einer Transition zu ändern. Erlaubt sind hierbei jedoch nur sinnvolle Konvertierungen, nicht also eine Umwandlung eines Zustands in eine Transition, siehe Abschnitt 4.17.

Zusammenfassend beschrieben nimmt die Methode `MyGraphModel.convert(DefaultGraphCell, Class)` ein *JGraph*-Diagrammelement, welches ein Zustand oder eine Transition sein kann. Die Methode erzeugt ein neues *JGraph*-Element gemäß dem `Class`-Parameter. Hiernach kopiert sie alle Eigenschaften vom ursprünglichen Element in das neu erzeugte Diagrammelement. Dann löscht sie das ursprüngliche Element und fügt das neue Element in das vorhandene *JGraph*-Datenmodell ein.

Bei der Änderung des Datenmodells aufgrund der Ersetzung des alten Elements durch das neue ist zu beachten, dass sowohl das neue Element an den Vater des alten Elements gebunden wird als auch der Vater die Referenz zu dem alten Element verliert. Gleiches gilt sinngemäß für die Kinder des alten Elements, welche Zustände und Transitionen sein können. Diese müssen ebenso umgehängt werden auf das neue Element. Des Weiteren müssen all diejenigen Transitionen bearbeitet werden, bei denen das alte Element den Quellzustand oder Zielzustand darstellt.

An jedem `GraphCell`-Objekt, also jedem *JGraph-Model*-Diagrammelement, hängt als weitere Eigenschaft das korrespondierende *KIEL*-Diagrammelement. Bei Erzeugung eines neuen *JGraph*-Elements muß auch ein neues *KIEL*-Element erzeugt werden, welches an das neue *JGraph*-Element gehängt wird. Auch hier müssen alle Eigenschaften des alten *KIEL*-Elements nach dem neuen *KIEL*-Element kopiert werden. Hierzu gehört neben Eigenschaften wie *Events*, Variablen und Beschriftungen auch das Löschen des alten *KIEL*-Elements aus dem *KIEL*-Datenmodell und das Einfügen des neuen *KIEL*-Elements in diese Datenstruktur. Bei der Behandlung der Layoutinformationen im *KIEL*-Datenmodell ist ferner zu beachten, dass diese bei einem Typ-Morphing zwar nicht neu angelegt werden müssen, dass aber die vorhandene Layoutinformation im *KIEL*-Datenmodell an das neu erzeugte Element gebunden werden muss.

Hinzufügen einer *Delimiter-Line*

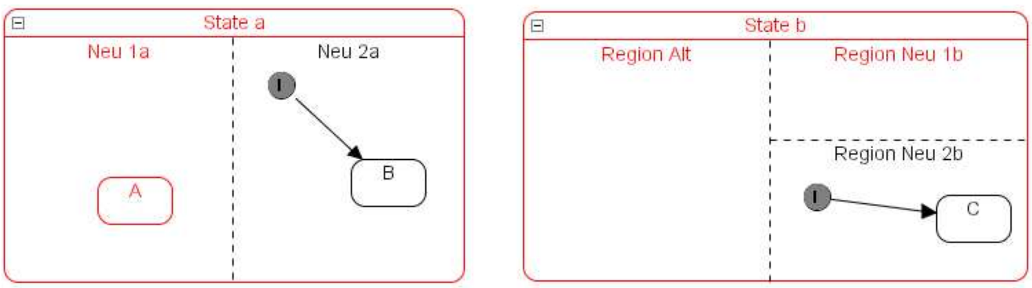


Abbildung 5.8.: Hinzufügen einer *Delimiter-Line*

Dies ist die aufwändigste Datenmodell-Änderung. Die Methode `MyGraphModel-.addDelimiter(Point2D start, Point2D end)` fügt an der angegebenen Koordinatenposition eine *Delimiter-Line* hinzu. Die folgenden Ausführungen werden in Abbildung 5.8 illustiert und beziehen sich auf das Hinzufügen der *Delimiter-Line* des linken *Composite-States* und auf das Hinzufügen der horizontalen *Delimiter-Line* des rechten *Composite-States*. Zunächst ist zu prüfen, ob die angegebene Position innerhalb eines *Composite-States* liegt und ob es sich nicht um den *Root-State* handelt. Ist der ermittelte Zustand ein *OR-State* (in der Abbildung 5.8 der like *Composite-State*), so wird zunächst eine Konvertierung in einen *AND-State* durchgeführt, siehe Abschnitte 4.17 und 5.3.3. Sofern es sich nun um einen *AND-State* handelt, werden zwei *Regions* (in der Abbildung bezeichnet mit *Neu 1a* und *Neu 2a*) erzeugt, andernfalls (in der Abbildung 5.8 der rechte *Composite-State*) handelt es sich um eine *Region* (*Region Neu 1b*) und diese wird zweigeteilt, es wird also in diesem Fall nur *eine* neue *Region* (*Region Neu 2b*) erzeugt. In beiden Fällen besteht die Aufgabe nun darin, zum einen die Koordinaten beider *Regions* zu bestimmen und zu setzen und zum anderen Kinder in der Hierarchie zu verschieben. Die neuen Koordinaten der *Regions* hängen davon ab, ob es sich um eine vertikale oder horizontale *Delimiter-Line* handelt, und welche räumliche Ausdehnung der *AND-State* hat. Wird nur *eine Region* neu erzeugt, so müssen diejenigen Kinder in die neue *Region* hierarchisch verschoben werden, welche in dem räumlichen Gebiet der neuen *Region* liegen (in der Abbildung 5.8 Zustand *C* und der *Initial-State*). Werden hingegen *zwei Regions* erzeugt, der *AND-State* also auf zwei *Regions* aufgeteilt, so müssen *alle* Kinder des *AND-States* je nach ihrer Lage hierarchisch in die beiden *Regions* verschoben werden (Zustände *A* und *B* sowie der *Initial-State*. Im letzten Schritt werden die neu erzeugten *Regions* mitsamt der neu erzeugten *Delimiter-Line* als Kinder dem *AND-State* hinzugefügt.

Löschen einer *Delimiter-Line*

Zunächst müssen diejenigen *Regions* ermittelt werden, welche durch das Löschen der *Delimiter-Line* zusammengeführt werden sollen. Sofern es sich um eine horizontale *Delimiter-Line* handelt (z. B. in der Abbildung 5.8 im rechten Zustand), müssen die x-Koordinaten von *Region* und *Delimiter-Line* gleich sein, beide müssen die gleiche Länge aufweisen, und entweder liegt die obere oder die untere Kante der *Region* auf der *Delimiter-Line*. Sinngemäß muss die Prüfung erfolgen, wenn eine vertikale *Delimiter-Line* gelöscht werden soll. Bei diesem Vergleich von Koordinatenwerten und Längen- und Breitenangaben ist zu berücksichtigen, dass die *Regions* nicht exakt den gesamten Platz eines *AND-States* ausfüllen müssen und ihre Kanten auch nicht exakt auf *Delimiter-Lines* liegen müssen. Dies ist bei *Regions* nur dann der Fall, wenn diese durch das Hinzufügen einer *Delimiter-Line* im Editor erzeugt wurden. In anderen Fällen, z. B. nach Anwendung des *KIEL*-Layouters oder durch Import eines *Statecharts* über das *KIEL-File-Interface* können Ungenauigkeiten auftreten. Als maximal tolerierte Ungenauigkeit werden zehn Pixel im hier vorgestellten Algorithmus fest definiert.

Nachdem die beiden *Regions* gefunden sind, werden die Kinder der unteren oder rechten von beiden *Regions* hierarchisch verschoben in die obere beziehungsweise linke *Region*. Hiernach wird erstere *Region* gelöscht und die Koordinaten der oberen beziehungsweise linken *Region* so angepasst, dass diese *Region* nun den Platz beider *Regions* ausfüllt. Wurde die einzige *Delimiter-Line* des *AND-State* gelöscht, so wird dieser *AND-State* in einen *OR-State* konvertiert, siehe hierzu die Abschnitte 4.17 und 5.3.3.

Hinzufügen eines Zustands

Es werden ein *KIEL*-Diagrammelement und die zugehörige Layout-Information erzeugt und sodann für diesen Zustand mittels der `MyGraphModel.add(Node)`-Methode, siehe Abschnitt 5.3.2, ein entsprechendes *JGraph*-Element generiert. Hiernach wird der Vater dieses Elements ermittelt, indem nach dem kleinsten *Composite-State* gesucht wird, welcher die Fläche des neuen Zustands vollständig umfasst.

Hinzufügen einer Transition

Im ersten Schritt werden Quellzustand und Zielzustand ermittelt. Falls der Quellzustand ein *Initial-State* ist, wird der Typ der neuen Transition ein *Initial-Arc* sein, im Falle eines *Dynamic-Choice* wird es eine *Conditional-Transition* sein und falls der Quellzustand ein *Suspend* ist, wird eine *Suspension-Transition* erzeugt werden. Falls jedoch der Quellzustand keinem der vorgenannten Typen entspricht, so wird eine Transition gemäß der Auswahlbox in der Werkzeugleiste der graphischen Benutzeroberfläche erzeugt.

Als nächstes wird die niedrigste Priorität einer von diesem Zustand ausgehenden Transition ermittelt, um der neuen Transition eine noch um eins niedrigere Priorität zu geben.

Sofern es sich bei der neu anzulegenden Transition um eine Loop-Transition handelt, müssen der Transition zwei Stützpunkte mitgegeben werden, um eine kreisförmige Ausdehnung vom Startpunkt zum Zielpunkt zu erreichen, wobei Start- und Zielpunkt identisch sein werden. Die Art des Linienzugs – zur Auswahl stehen Bezier-Kurve, Spline-Kurve und Polygon – wird entsprechend der Einstellung in der Auswahlbox der Werkzeugleiste gesetzt.

Zum Erzeugungsprozess gehört des Weiteren eine Überprüfung, ob die zu erzeugende Transition nach den syntaktischen Regeln erlaubt ist. Dies ist nicht der Fall, wenn der Großvater des Quellzustands ungleich dem Großvater des Zielzustands ist – hierbei würde es sich um eine Interleveltransition handeln – oder wenn Quell- oder Zielzustand nicht als solcher vorgesehen ist. Alle *Final-States* und alle *History-States* sind als Quellzustand nicht erlaubt, von ihnen dürfen also keine Transitionen abgehen. Alle *History-States*, *Initial-States*, *Suspends* und der *Root-State* kommen als Zielzustand einer Transition nicht in Frage.

Kollabieren und Expandieren eines *Composite-States*

Das *JGraph*-Datenmodell wird um die Attribute `isCollapsed` und `expandedBounds` erweitert. Beim Kollabieren eines *Composite-States* wird das `isCollapsed`-Attribut gesetzt, und die aktuellen räumlichen Grenzen des *Composite-States* werden im `expandedBounds`-Attribut gesichert. Mittels gleitendender Strukturveränderung (siehe Abschnitt 4.7), erhält der kollabierende Zustand dann die Standardbreite und Standardhöhe, welche auch bei Erzeugung von *OR-States* Anwendung finden. Hiernach wird die Sichtbarkeit aller Kinder dieses Zustands aufgehoben.

Das Expandieren erfolgt analog, jedoch mit dem Unterschied, dass eine gleitende Strukturveränderung in Richtung der zuletzt gespeicherten `expandedBounds` durchgeführt wird. Daneben ist beim Expandieren zu beachten, dass die Positionsangaben aller Diagrammelemente im *JGraph*-Modell relativ zum Ursprung des Koordinatensystems und nicht relativ zum Vater-Element gesetzt sind. Grundsätzlich werden durch das Rahmenwerk nur die Koordinaten der sichtbaren Diagrammelemente automatisch beim Verschieben geändert. Wird jedoch ein kollabierter Zustand verschoben und später wieder expandiert, so müssen das räumliche Delta der Verschiebung errechnet und die Koordinaten aller Kinder um dieses Delta verändert werden.

5.3.4. Editoreigene Layoutmechanismen

Es wird ein Algorithmus implementiert, welcher dem Benutzer die Aufgabe abnimmt, während des Platzierens eines neuen Diagrammelements oder vor dem Expandieren eines Zustands zunächst andere Diagrammelemente zu verschieben, um auf diese Weise für die nächste Aktion Platz zu schaffen. Unter Platzieren wird hier verstanden das Wählen einer Position und das nachfolgende Verschieben, bis die Maustaste gelöst wird und der Zustand seine endgültige Position erhalten hat.

Sowohl beim Expandieren als auch beim Platzieren eines Zustands wird die Methode `MorphingLayouter.addCells(DefaultGraphCell, Rectangle2D)` aufgerufen mit dem Diagrammelement, welches entweder platziert oder expandiert werden soll (erster Parameter) sowie mit den neuen Grenzen dieses Diagrammelements (zweiter Parameter). Es wird eine Menge von Diagrammelementen ermittelt, welche aufgrund ihrer Lage zu den neuen Grenzen dieses Diagrammelements verschoben oder vergrößert werden müssen. Beim Platzieren oder Expandieren eines neuen Zustands werden der umgebende Zustand vergrößert und, soweit notwendig, alle rechts und unterhalb liegenden Elemente weiter nach rechts und unten verschoben.

Zunächst wird geprüft, ob die neue Fläche des expandierenden Zustands größer als seine alte Fläche ist. Ein neu anzulegender Zustand wird ebenfalls als zu expandierender Zustand betrachtet. Betrachtet werden nun all diejenigen Zustände des *Statecharts*, welche mit dem expandierenden Zustand einen gemeinsamen Vater haben oder Kinder eines solchen Zustands sind, gemeint sind also Geschwister und ihre Kinder. Für jeden dieser Zustände wird geprüft, ob er rechts oder unter dem expandierenden Zustand liegt und ob er den Zustand schneiden wird, was auch

abhängig von der Benutzereinstellung `morphingLayouterBorderDistance` (siehe Abschnitt 4.22) ist. Sind beide Eigenschaften erfüllt, so wird dieser Zustand entsprechend verschoben. Anhand der neuen Position des Zustands wird sodann überprüft, ob dieser den Vater des expandierenden Zustands schneidet. Ist dies der Fall, so wird dieser Vater entsprechend vergrößert. Rekursiv wird dieses Verfahren dann angewendet auf jedes durch diesen Algorithmus veränderte Element.

5.3.5. Layout-Änderungen

Im Standard-Modus, siehe Abschnitt 5.2.3, kann der Benutzer Größen- und Positionsänderungen an Diagrammelementen durchführen. Nach jeder solcher Änderungen werden die Vater-Kind-Beziehungen auf der Basis des aktuellen Layouts aktualisiert. Dies hat zur Folge, dass Zustände in der Hierarchie ihre Position ändern, falls diese Zustände z. B. durch Vergrößerung eines *OR-States* in den Bereich dieses *OR-States* fallen. Auf diese Weise kann also um eine Menge von Zuständen ein neuer Zustand gelegt werden und damit diese Zustände in der Hierarchie nach unten verschoben werden. Ebenso ist es möglich, einen Zustand aus seinem Vater herauszuziehen. Hierdurch erhält der Zustand einen neuen Vater und ändert somit ebenso seine Position in der Hierarchie.

5.3.6. Layoutgestützte gleitende Strukturveränderung

Eine gleitende Strukturveränderung besteht, wie in Abschnitt 4.7 beschrieben, aus mehreren Zwischenanordnungen. Im Folgenden wird beschrieben, wie diese einzelnen Zwischen-Anordnungen berechnet werden und wie die Frequenz der Anordnungen unabhängig von der Rechen- und Darstellungskapazität der Hardware konstant gehalten wird. Anders als *SVG*-Rahmenwerke bietet *JGraph* keine Animationen, daher muss ein solches Verhalten emuliert werden.

Auf einer Kopie des *JGraph*-Datenmodells des Ausgangsdiagramms wird die vollständige Änderung durchgeführt. Sodann wird für jeden Punkt, welcher sich ändern soll, der Änderungsbetrag in Pixeln errechnet. Beispielsweise werden bei einer Positionsänderung eines Zustands die x- und y-Koordinate der Ursprungsposition mit den jeweiligen Koordinaten der Zielposition verglichen. Bei Transitionen wird dies für alle Stützpunkte durchgeführt.

Der hier entwickelte Algorithmus sieht vor, dass die Anzahl n der Anordnungszwischenschritte der größten hierbei ermittelten Änderung in Pixeln entspricht. Hieraus folgt, dass in jedem Änderungsschritt jeder Punkt höchstens um einen Pixel geändert wird und im Allgemeinen um einen Bruchteil eines Pixels.

Allgemein folgt für das Delta D_k für jede Koordinate k als Änderung für jeden Schritt

$D_k = (k_{ziel} - k_{original}) / n$

Rundungsfehler werden umgangen, indem alle Koordinaten als Fließkommazahlen gespeichert werden.

Nachdem D_k für alle zu ändernden Koordinaten berechnet ist, können iterativ in n Schritten Änderungen am Diagramm durchgeführt werden. Eine gleitende Strukturveränderung kann somit aus mehreren tausend Diagrammänderungen bestehen.

Die Frequenz der Änderungsschritte ist zunächst abhängig von der zur Verfügung stehenden Rechenleistung. Denn einerseits kann jede dieser Diagrammänderungen ihrerseits je nach Komplexität des Diagramms aus vielen Berechnungsschritten bestehen. Andererseits ist auch das Darstellen der graphischen Information zeitintensiv. Um die Abhängigkeit der Frequenz der Änderungsschritte durch den Benutzer festlegen lassen zu können, muss ein Regelmechanismus hinzugefügt werden, der eine bestimmte Frequenz sicherstellt. Eine Zielfrequenz von 50 Hertz bedeutet, dass jede sich ändernde Koordinate sich um höchstens 50 Pixel pro Sekunde ändert.

Dieser Regelmechanismus muss die tatsächlich erreichte Frequenz verringern und erhöhen können, um die Zielfrequenz erreichen zu können. Eine Verringerung der Frequenz wird durch Pausen im Programmablauf erreicht, eine Erhöhung der Frequenz durch Zusammenfassen mehrerer Veränderungsschritte. Werden m solcher Schritte zusammengefasst, ändert sich jede Koordinate um höchstens m Pixel in einem zusammengefassten Schritt, *eine* Koordinate ändert sich jedoch mindestens um genau m Pixel. Dies folgt aus den obigen Voraussetzungen. Muss der Regelmechanismus die Frequenz erhöhen, resultieren daraus größere Sprünge in der Veränderung des Diagramms, da die Koordinaten sich nun nicht mehr in Ein-Pixel-Schritten ändern, sondern in gröberen Weiten.

Ob eine Regelungsmaßnahme notwendig ist, wird periodisch geprüft, indem die tatsächlich verbrauchte Zeit für einen Veränderungsschritt gemessen und mit dem Sollwert – bei 50 Hertz wären das 20 Millisekunden – verglichen wird. Natürlich muss dieses Regelsystem Toleranzen erlauben: Hat ein Veränderungsschritt z. B. statt der geforderten 20 Millisekunden 21 Millisekunden gebraucht, würde eine Erhöhung der aktuellen Frequenz, also eine neue Schrittweite von zwei, dazu führen, dass die folgenden Veränderungsschritte nur noch 10,5 Millisekunden benötigen. Außerdem muss berücksichtigt werden, dass die zur Verfügung stehenden Rechen- und Darstellungsressourcen sich von Iteration zu Iteration verändern können und vom Editor nicht beeinflusst werden können.

Sinnvoll erscheint hier der Ansatz, nicht jeden einzelnen Veränderungsschritt zu messen, sondern Gruppen von etwa 10 oder 20 Schritten. Hierbei fällt die sich ständig ändernde Rechenleistung nicht so stark ins Gewicht, vielmehr wirkt sie sich als Durchschnittswert der letzten 10 oder 20 Schritte aus.

5.3.7. Nebenläufigkeit

Eine gleitende Strukturveränderung, siehe Abschnitt 4.7, wird im Editor immer vom Benutzer angestoßen, also im *AWT-Event-Thread* der *JVM* initiiert. Dieser *Thread* ist allerdings höher priorisiert als der *System-Thread* der *JVM*, welcher die graphische Oberfläche aktualisiert. Liefe nun eine gleitende Strukturveränderung im *AWT-Event-Thread*, so würde die Oberfläche erst dann neu gezeichnet werden,

wenn die Strukturveränderung bereits abgearbeitet ist. Die Folge wäre, dass nur das letzte Bild dieser Strukturveränderung angezeigt würde, für den Benutzer also diese gleitende Strukturveränderung nicht sichtbar wäre.

Aus diesem Grunde sollen langlaufende Aktionen, insbesondere aufwändige Graphik-Routinen, nicht in diesem *AWT-Event-Thread* der *JVM* ausgeführt werden. Die gleitende Strukturveränderung muss daher in einen *Thread* ausgelagert werden und im Hintergrund laufen. Problematisch hierbei ist allerdings, dass eine gleitende Strukturveränderung vom Benutzer angestoßen werden kann, auch wenn solch eine Aktion gerade läuft. Würden nun diese Aktionen sofort in einem neuen *Thread* ausgeführt werden, so würden Interferenzen auftreten und das Resultat wäre unvorhersehbar. Tatsächlich möchte der Benutzer allerdings eine gleitende Strukturveränderung immer ausführen auf dem Resultat der vorhergehenden. Folglich müssen die gleitenden Strukturveränderungen in eine Abarbeitungsschlange aufgenommen werden und sukzessive abgearbeitet werden. Diese Abarbeitungsschlange wird implementiert durch einen *Thread*, welcher permanent läuft, um Abarbeitungsanfragen entgegenzunehmen.

5.3.8. Bearbeitungsschritt-Historie und *Undo-and-Redo*-Funktionalität

Diese Funktionsmerkmale werden in den Abschnitten 4.9 und 4.14 dargestellt. Wie in Abschnitt 5.2.3 erläutert, werden alle Benutzeraktionen als Objekt repräsentiert. Jeder Benutzer-Bearbeitungsschritt, welcher sich auf das Editieren eines *Statecharts*bezieht, umfasst im Allgemeinen mehrere Datenmodell-Änderungsschritte. Wird z. B. ein Zustand kollabiert, dann ändert sich mit diesem Benutzer-Bearbeitungsschritt die Sichtbarkeit der Kind-Elemente sowie die Größe des kollabierenden Zustands.

Die grundlegende Funktionalität der *Undo-and-Redo*-Funktionen wird durch das *JGraph*-Rahmenwerk bereitgestellt. Die dort implementierten Funktionen basieren allerdings nicht auf Bearbeitungsschritten sondern auf *Model*-Änderungen. Ein Bearbeitungsschritt kann mehrere *Model*-Änderungen nach sich ziehen. Deswegen muss die vom Rahmenwerk bereitgestellte Funktionalität um einen Cluster-Mechanismus erweitert werden. Dieser Mechanismus ermittelt die Anzahl n aller *Model*-Änderungen, welche seit Beginn eines Bearbeitungsschrittes durchgeführt worden sind. Dafür notwendig ist das Setzen von Transaktionsklammern, innerhalb derer gezählt wird. Ein vom Benutzer angestoßener *Undo*-Schritt wird n mal das vom Rahmenwerk bereitgestellte *Undo* ausführen. Analog wird bei *Redo* verfahren. Es werden also die *Model*-Änderungen gruppiert nach Bearbeitungsschritten. Die Bearbeitungsschritte werden gespeichert in *Stack*-Datenstrukturen, eine für Bearbeitungsschritte, welche rückgängig gemacht werden können (engl. *Undo*), und eine für Bearbeitungsschritte, welche schon rückgängig gemacht wurden und die erneut ausgeführt werden können (engl. *Redo*). Auf dem *Stack* wird dann je Bearbeitungsschritt die Anzahl n der *Model*-Änderungsschritte abgelegt.

6. Schlussbemerkungen

In den nachfolgenden Abschnitten werden zum einen die Ergebnisse der Implementierung dargestellt, wie sie im Abschnitt 6.1 des vorangegangenen Kapitels beschrieben wurde. Zum anderen wird ein Ausblick auf Erweiterungsmöglichkeiten der vorliegenden Arbeit in Form zusätzlicher Funktionalitäten gegeben. Hierbei handelt es sich überwiegend um weitere innovative Funktionsmerkmale, die bisher in verfügbaren Diagramm-Editoren nicht zu finden sind.

6.1. Ergebnisse

Zur Einordnung des Ergebnisses dieser Arbeit tragen eine Betrachtung des Programm-Quelltext-Umfangs, der Stabilität und des Funktionsumfangs bei.

Die Implementierung des Editors umfasst zusammen mit der des *KIEL*-Anwendungsfensters etwa 18.000 Programm-Quelltext-Zeilen, siehe im Anhang den Abschnitt A.4. Kompiliert und komprimiert entspricht dies etwa 295 kByte Maschinencode. Im Vergleich hierzu besteht das verwendete Framework *JGraph* aus etwa 143 kByte Maschinencode, der *KIEL*-Simulator aus 111 kByte, der *KIEL*-Browser aus 73 kByte, der *KIEL*-Layouter aus 59 kByte zuzüglich 16 kByte *C*-Code, und das *KIEL-Esterel-Studio*-File-Interface aus 51 kByte Maschinencode.

Bei einem ersten Systemtest und Gesamtintegrationstest ist erwartungsgemäß etwa ein Fehler je 1000 Programm-Quelltextzeilen in der Editor-Komponente aufgetreten. Diese Fehler werden im Rahmen dieser Arbeit behoben, um auch einen produktiven Einsatz des Editor-Moduls zu ermöglichen.

Der *KIEL*-Editor beinhaltet weit mehr Funktionalitäten als ursprünglich gefordert und geplant, vergleiche hierzu die von *KIEL* geforderten Merkmale in Abschnitt 1.2 und die tatsächlich implementierten Funktionalitäten, beschrieben in Kapitel 4. Es sind innovative Funktionsmerkmale enthalten, deren Anwendbarkeit und Nutzen in einer nachfolgenden Phase evaluiert werden können.

Bei ersten Anwendertests stellte sich heraus, dass auch ohne Benutzerhandbuch der Editor nach einer Einarbeitungsphase intuitiv genutzt werden kann. Sämtliche Funktionsmerkmale sind einsetzbar. Schwierigkeiten bereitet das Beschriften von Transitionen bei fehlender Kenntnis der vom Editor erwarteten Eingabesyntax. Hier besteht Verbesserungsbedarf, siehe Abschnitt 6.2. Die Anwendertests zeigten darüber hinaus, dass einige Konfigurationseinstellungen die Bedienbarkeit erheblich beeinflussen und sich ihre Voreinstellungen teilweise nur für geübte Anwender als günstig und geeignet erwiesen. Solche Einstellungen sind z. B. die Sensitivität bei der Selektion von Diagrammelementen, die Sichtbarkeit von *Size-Handles* und

die Größe der unsichtbaren Transitionsende-Verknüpfungspunkte. Als nachteilig stellt sich des Weiteren heraus, dass *AND-States* bislang nicht größenänderbar und *Delimiter-Lines* nicht verschiebbar sind. Der Grund hierfür ist, dass das Berechnen der *Region*-Größen nach einer *AND-State*-Größenänderung eine komplexe Aufgabe darstellt, welche vom *KIEL*-Layouter in Zukunft übernommen werden soll. Daher exisitiert diese Limitierung zur Zeit im Bereich des Größenänderns von Diagrammelementen. Darüber hinaus exisitieren keine ungelösten Probleme.

Abbildung 6.1 zeigt die Benutzeroberfläche des *KIEL*-Editors. *KIEL* soll sich neben seinem Funktionsumfang durch benutzerfreundliches, übersichtliches und intuitives Design der Benutzeroberfläche mit seinem Farbschema, welches durch ein eigenes *Look-And-Feel* [53] realisiert wird, auszeichnen.

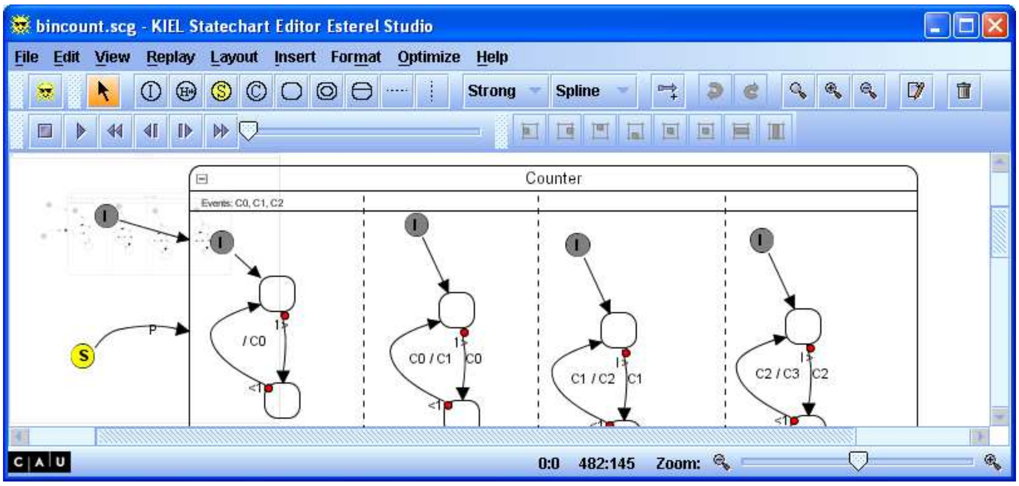


Abbildung 6.1.: *KIEL* im Editor-Modus

6.2. Ausblick

Im Folgenden werden Funktionalitäten beschrieben, deren Implementierung und Einsatz als sinnvolle Ergänzung zum bestehenden Funktionsumfang gesehen wird.

6.2.1. Multidimensionale *Undo-and-Redo*-Historie

Die aus Editoren bekannte *Undo-and-Redo*-Funktionalität stellt eine eindimensionale Historie der durchgeführten Bearbeitungsschritte dar. Der Benutzer kann n Bearbeitungsschritte rückgängig machen und sich danach entscheiden, ob er die n Bearbeitungsschritte endgültig verwerfen möchte, indem er nun einen neuen Bearbeitungsschritt ausführt, oder ob er die letzten $m <= n$ rückgängig gemachten Schritte doch nicht rückgängig machen möchte, er also weniger weit in die Bearbeitungsvergangenheit zurück gehen möchte. Entscheidend hierbei ist, dass

bei dem bekannten Verfahren *Redo*-Schritte nur im direkten Anschluss an *Undo*-Schritte möglich sind. Sobald nach *Undo*-Schritten ein neuer Bearbeitungsschritt ausgeführt wird, sind keine *Redo*-Schritte mehr verfügbar, der vormals gegangene Bearbeitungsweg ist endgültig verloren.

Eindimensional ist diese Historie, weil Bearbeitungswege verloren gehen und sich der Benutzer somit nur auf einem linearen Bearbeitungsweg vorwärts und rückwärts bewegen kann. Eine sinnvolle Ergänzung stellt die Erweiterung der *Undo-and-Redo*-Technik um eine Versionsverwaltung mit integrierter Bearbeitungsschritthistorie dar: Sobald nach *Undo*-Schritten ein neuer Bearbeitungsschritt ausgeführt werden soll und damit die verworfenen Bearbeitungsschritte entgültig verloren gehen, wird die aktuelle Version, also der zu verwerfende Weg, gespeichert mitsamt seiner Bearbeitungsschritthistorie. Auf diese Weise geht kein Bearbeitungsschritt verloren und kann wiederhergestellt werden, sofern der vorhergehende Bearbeitungsschritt wiederhergestellt wurde.

Beispiel

Der Benutzer führt zehn Bearbeitungsschritte aus. Hiernach entscheidet er sich, die letzten drei Schritte zu verwerfen mittels *Undo*. Er gelangt also wieder zurück nach Bearbeitungsschritt sieben. An dieser Stelle schlägt er einen neuen Bearbeitungsweg ein und geht vier neue Schritte. Er stellt nun fest, dass der neu eingeschlagene Bearbeitungsweg auch ungünstig war, verwirft die vier Schritte und steht nun wieder an Bearbeitungsschritt sieben. Hier kann er sich nun entscheiden, ob er einen weiteren neuen Weg einschlägt oder er die verworfenen alten drei Schritte oder die neueren vier Schritte wiederherstellt. Statt eines linearen Bearbeitungswegs entsteht eine baumförmige Struktur. An Bearbeitungsschritt 7 z. B. gibt es nun eine Verzweigung in zwei fortführende Richtungen, zu der beliebig viele weitere Verzweigungen (Versionen) hinzukommen können.

6.2.2. Speichern der *Undo-and-Redo*-Historie

In einem Editor, welcher eine *Undo-and-Redo*-Technik anbietet, wächst die *Undo-and-Redo*-Historie der verwerfbaren Bearbeitungsschritte mit jedem Bearbeitungsschritt. Zu jedem Zeitpunkt kann ein Bearbeitungsschritt zurückgenommen werden. Wird jedoch der Editor nach Speicherung der Arbeit verlassen und danach erneut gestartet, so ist die vormals aufgebaute Bearbeitungsschritt-Historie nicht mehr vorhanden. Es können grundsätzlich nur Bearbeitungsschritte verworfen werden, die in der aktuellen Sitzung erzeugt wurden. Wünschenswert wäre allerdings, dass neben dem Speichern der Arbeit auch die Historie gesichert wird. Da solch eine Funktionalität eine gewisse Menge an Speicherkapazität verbraucht, könnte das Speichern der Historie auf z. B. diejenigen Arbeiten beschränkt werden, welche im Menü *Files*→*Last-Files* zugreifbar sind.

6.2.3. Bearbeitung von Transitionsbeschriftungen

Bei einem ersten Anwendertest hat sich herausgestellt, dass der Editor das Editieren von Transitionsbeschriftungen unterstützen könnte, indem bei der Eingabe so wenig syntaktische Regeln wie möglich beachtet werden müssen. Beispielsweise könnte, wie in *Esterel-Studio*, die Eingabe von *Trigger*, *Condition* und Action auf drei separate Eingabefelder aufgeteilt werden. Des Weiteren könnte farblich während der Eingabe kenntlich gemacht werden, ob / wann die Eingabe korrekt ist.

6.2.4. Erweitertes Diagrammelementtyp-Morphing

Bei dem im Editor implementierten Diagrammelementtyp-Morphing, siehe Abschnitt 4.17, kann der Benutzer explizit den Typ eines Diagrammelements ändern, z. B. kann unter anderem ein *Simple-State* in einen *OR-State* verwandelt werden. Neben diesen expliziten Typänderungen stellen die impliziten Änderungen eine ebenso hilfreiche Erweiterung dar. Hierunter wird verstanden das Umwandeln eines *OR-States* in einen *AND-State*, sobald der Benutzer eine *Delimiter-Line* hinzufügt. Die Umwandlung geschieht dann automatisch im Hintergrund als zusätzliche und syntaktisch notwendige Operation. In der Gruppe der impliziten Typumwandlungen wäre des Weiteren sinnvoll die Änderung von *Simple-State* nach *OR-State* oder *AND-State*, sobald in den *Simple-State* Zustände eingefügt werden. Gleiches gilt für das Entfernen aller Unterzustände eines *Composite-States*. Zudem könnte ein *Final-State* in einen *Non-Final-State* umgewandelt werden, sobald er eine ausgehende Transition erhält.

6.2.5. Erweiterter Zoom und Ausschnittsänderung

Mittels Zoom wird im *KIEL*-Editor die Ansichtsdarstellung eines *Statecharts* vergrößert oder verkleinert. Die tatsächlichen Koordinaten der Diagrammelemente bleiben unverändert. Vorstellbar wäre demgegenüber auch, das Diagramm in seiner tatsächlichen graphischen Repräsentation zu vergrößern oder zu verkleinern, indem seine Koordinaten verändert werden.

Die Änderung des sichtbaren *Statechart*-Ausschnitts soll nicht nur mittels Scrollleisten, Tasten und dem Mausrad erfolgen können, sondern es soll auf eine geeignete Weise die Übersichtsansicht in den Vordergrund treten können, in der dann durch Zeichnen eines Rechtecks der neue sichtbare Ausschnitt für die Bearbeitungssicht gewählt werden kann.

6.2.6. Layouthilfen während der Bearbeitung

In Abschnitt 4.13 wird eine Layouthilfe beschrieben, welche den Benutzer während des Anlegens eines neuen Zustands unterstützt, indem das Layout des Diagramms während des Platzierens des neuen Elements derart verändert wird, dass genügend Platz für das einzufügende Diagrammelement zur Verfügung steht. Denkbar ist,

solche Layouthilfen auch während einer Verschiebe- oder Vergrößerungsoperation anzubieten. Soll ein Zustand verschoben werden, muss der Benutzer angeben, ob der zu verschiebende Zustand seinen Platz in der *Statechart*-Hierarchie ändern soll, oder er innerhalb seines Vaters an einer anderen Stelle platziert werden soll. In letzterem Fall sollte der Vater bei Bedarf automatisch vergrößert werden. Eine gleichzeitige notwendige Positionsänderung der Geschwister ist dabei auch denkbar. Möchte der Benutzer dagegen einen Zustand vergrößern und mit dieser Aktion *nicht* die Position anderer Zustände in der *Statechart*-Hierarchie ändern, indem sie durch Vergrößerung des Zustands Kinder desselben werden, so kann diese Aktion mit ähnlichen Mechanismen des *Platz-Schaffens* vom Editor unterstützt werden.

Grundsätzlich sollten solche Layouthilfen wie die hier beschriebenen visuell unterstützt werden durch Verwendung verschiedener Mauszeiger, welche den Benutzer darüber informieren, welche editoreigenen Mechanismen den aktuellen Bearbeitungsschritt begleiten.

7. Literaturverzeichnis

[1] Gaudenz Alder. Design and Implementation of the JGraph Swing Component. http://jgraph.com/paper.html.

[2] Gaudenz Alder. JGraph Diagram Component, 2001. URL `http://sourceforge.net/projects/jgraph`.

[3] ArgoUML. The ArgoUML project. URL `http://argouml.tigris.org/`. An open source UML design tool.

[4] Benjamin B. Bederson, Jesse Grosjean und Jon Meyer. Toolkit Design for Interactive Structured Graphics. *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, 30(8), August 2004.

[5] Gerard Berry. *The Esterel v5 Language Primer*. Draft Book, 1999. URL `ftp://ftp-sop.inria.fr/esterel/pub/papers/primer.ps`.

[6] Christian Lenz Cesar. Graph Foundation Classes for Java, 1999. URL `http://www.alphaworks.ibm.com/tech/gfc`.

[7] Charles André. Semantics of S.S.M (Safe State Machine). Technischer bericht, I3S, Sophia-Antipolis, France, 2003. URL `http://www.esterel-technologies.com/v3/?id=50399\&dwnID=48`.

[8] Rowan Christmas. Graph Interface library, 2002. URL `http://sourceforge.net/projects/csbi`.

[9] D. Cox, J. S. Chugh, C. Gutwin und S. Greenberg. The Usability of Transparent Overview Layers. In *Companion Proceedings of the CHI '98 Conference on Human Factors in Computing Systems*. ACM Press, 1997.

[10] Reinhard Diestel. *Graphentheorie*. Springer-Verlag, 2000.

[11] Tim Dwyer und Peter Eckersley. WilmaScpoe Homepage, 2003. URL `http://www.wilmascope.org/`.

[12] Esterel Technologies. Company homepage. `http://www.esterel-technologies.com`.

[13] GEF Graph Editing Framework. Project Homepage, 2003. URL `http://gef.tigris.org/`.

[14] Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides. *Design Patterns*. Addison-Wesley, 1995.

[15] Gaudenz Alder. The JGraph Tutorial. `http://jgraph.com/tutorial.html`.

[16] Thomas Gehrke. *Dynamische Modelle für Reaktive Systeme mit Daten*. PhD thesis, Technische Universität Braunschweig, Dezember 2000.

[17] Les Grove, Wells Mathews, Rodney Hickman und Kal Toth. Open Source Software Engineering Tools. Aufsatz, Portland State University, 2004.

[18] Reinhard von Hanxleden. Graphical languages for modeling complex reactive systems. Workshop on Synchronous Languages SYNCHRON 2003, Luminy, France, Dezember 2003. `http://www-verimag.imag.fr/PEOPLE/Nicolas.Halbwachs/SYNCHRON03/Abstracts/vonhanxleden.html`.

[19] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, Juni 1987.

[20] Jon Harris. JGraphEd. URL `http://www.jharris.ca/JGraphEd/`.

[21] Andreas Hartmann. *Ein grafisches Werkzeug zur Unterstützung von Model-Checking-Prozessen*. Universität Rostock, 2002.

[22] John E. Hopcroft und Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, Reading, MA, 1979.

[23] Randy Hudson. Graphical editing framework, 2003. URL `http://www.eclipse.org/gef/`.

[24] Michael Jünger und Petra Mutzel. *Graph Drawing Software*. Springer, Oktober 2003. ISBN 3540008810.

[25] Jens Kanschik. HyperGraph, 2003. URL `http://sourceforge.net/projects/hypergraph`.

[26] Lutz Kirchner und Ulrich Frank. Evaluierung von UML-Modellierungswerkzeugen. Aufsatz, Universität Koblenz-Landau, 2004.

[27] Gerwin Klein. Generating Editors with Grace, 1999. URL `http://www.doclsf.de/grace/`.

[28] Tobias Kloss. Flexibles und Automatisiertes Layout von Statecharts. Studienarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, Juli 2003.

[29] Tobias Kloss. Automatisches Layout von Statecharts unter Verwendung von GraphViz. Diplomarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, Mai 2005.

[30] Philippe Kruchten. The Rational Unified Process An Introduction. Addison Wesley, Dezember 2003.

[31] M. S. Marshall, I. Herman und G. Melançon. An object-oriented design for graph visualization. *Software: Practice and Experience*, 31(8):739–756, 2001. URL citeseer.ist.psu.edu/article/marshall00objectoriented.html.

[32] M. Scott Marshall. GVF - Graph Visualization Framework, 2001. URL http://sourceforge.net/projects/gvf/.

[33] Mathworks Inc. Simulink Documentation, 2005. http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/.

[34] Mathworks Inc. Stateflow Application Programming Interface, 2005. http://www.mathworks.com/access/helpdesk/help/toolbox/stateflow/.

[35] Object Management Group. OMG Unified Modeling Language Specification, Version 1.5, März 2003. http://www.omg.org/cgi-bin/doc?formal/03-03-01.

[36] Bernd Oestereich. *Objektorientierte Softwareentwicklung: Analyse und Design mit der Unified modeling language*. Oldenbourg, München, Wien, 5 edition, 2001.

[37] André Ohlhoff. Simulating the Behavior of Synccharts. Studienarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, November 2004.

[38] Joshua O'Madadhain. Java Universal Network/Graph Framework, 2003. URL http://sourceforge.net/projects/jung.

[39] Steffen Prochnow und Reinhard von Hanxleden. Visualisierung komplexer reaktiver Systeme – Annotierte Bibliographie. Technischer Bericht 0406, Christian-Albrechts-Universität Kiel, Institut für Informatik und Praktische Mathematik, Juni 2004.

[40] Tom Sawyer. Tom Sawyer Visualization, Java Edition. URL http://www.tomsawyer.com.

[41] Alexander Shapiro. ToughGraph, 2001. URL http://www.touchgraph.com/.

[42] Frank Steinbrückner. Analyse von Software Systemen - Projekt Argo/UML. Seminar, Technische Universität Cottbus, 2002.

[43] Sun Microsystems, Inc. Code conventions for the java programming language. http://java.sun.com/docs/codeconv/.

[44] Jan Täubrich. Untersuchung der nicht-interaktiven Simulation von Stateflow-Statecharts. Praktikumsbericht, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, April 2005.

[45] The KIEL Project. Project API Documentation, 2004. URL http://www.informatik.uni-kiel.de/~rt-kiel/kiel/kiel/doc/. Kiel Integrated Environment for Layout.

[46] The KIEL Project. Project Homepage, 2004. URL http://www.informatik.uni-kiel.de/~rt-kiel/. Kiel Integrated Environment for Layout.

[47] The Mathworks, Inc. Company homepage. http://www.mathworks.com.

[48] Auburn University. Drawing Graphs with VGJ, 1998. URL http://www.eng.auburn.edu/department/cse/research/graph_drawing/graph_drawing.html.

[49] Dave Walend. JDigraph, 2001. URL http://sourceforge.net/projects/jdigraph.

[50] Mirko Wischer. Ein Browser für die Visualisierung dynamischer Sichten von Statecharts. Diplomarbeit, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, 2005.

[51] Mirko Wischer. Ein FileInterface für das KIEL Projekt. Praktikumsbericht, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, 2005.

[52] yWorks. yFiles Homepage, 2005. URL http://www.yworks.com/ger/products_yfiles_about.htm.

[53] John Zukowski. *John Zukowskiś Definite Guide to Swing for Java 2*. Apress, 1999.

.1. Funktionsumfang

.1.1. Allgemeine Editorfunktionen

Im Folgenden ist mit Konfigurierbarkeit gemeint, dass über den Menüeintrag *File→Preferences* ein bestimmtes Verhalten eingestellt werden kann.

- Erzeugung von *Statemate*- und *Esterel-Studio*-Modellen: In Menü *File→New* lässt sich zwischen diesen beiden Modellen wählen. Abhängig hiervon enthält die Werkzeugleiste entsprechende Diagrammelemente, siehe Abbildung .1.



(a) *Esterel-Studio*-Diagrammelemente (b) *Matlab*-Diagrammelemente

Abbildung .1.: Ausschnitt aus der Werkzeugleiste des Editors

- Exportformate *GIF* und *EPS*, wählbar über den Menüeintrag *File→Export*
- Ausdruck, wählbar über den Menüeintrag *File→Print*
- Konfigurierbarkeit von über 40 Eigenschaften über ein Dialogfenster, erreichbar über den Menüeintrag *File→Preferences*, siehe Abbildung .2 und die detaillierte Auflistung im Anhang .2.1.
- Fehlermeldungen sind selektiv unterdrückbar: Soll eine bestimmte Fehlermeldung in Zukunft nicht mehr angezeigt werden, so kann sie selektiv unterdrückt werden, siehe Abbildung .3. Diese Unterdrückung ist jederzeit wieder aufhebbar über den Menüeintrag *File→Message suppression reset*.
- Zwei Darstellungsschichten: Optional kann neben der Bearbeitungsschicht zusätzlich eine Übersichtsschicht eingeblendet werden, erreichbar über das Menü *View*, siehe Abschnitt 4.10.
- In beiden Darstellungsschichten wird das aktuell selektierte Diagrammelement hervorgehoben, siehe Abschnitt 4.10.
- Eine Gitternetzdarstellung ist optional einblendbar, aufrufbar in Menü *View*, siehe Abschnitt 4.8.
- *Undo-and-Redo*-Funktionalität, enthalten in Menü *Edit*
- Lokale Eventdeklarationen, Variablendeklarationen und *State-Actions* sind optional einblendbar über das Menü *View*.
- Replay: Bearbeitungsschritte können abgespielt werden in normalem und schnellem Tempo vorwärts und rückwärts; Mediaplayer-Optik und -Handhabung; Diese Funktionalität ist erreichbar über die Werkzeugleiste und das Menü *Replay*.

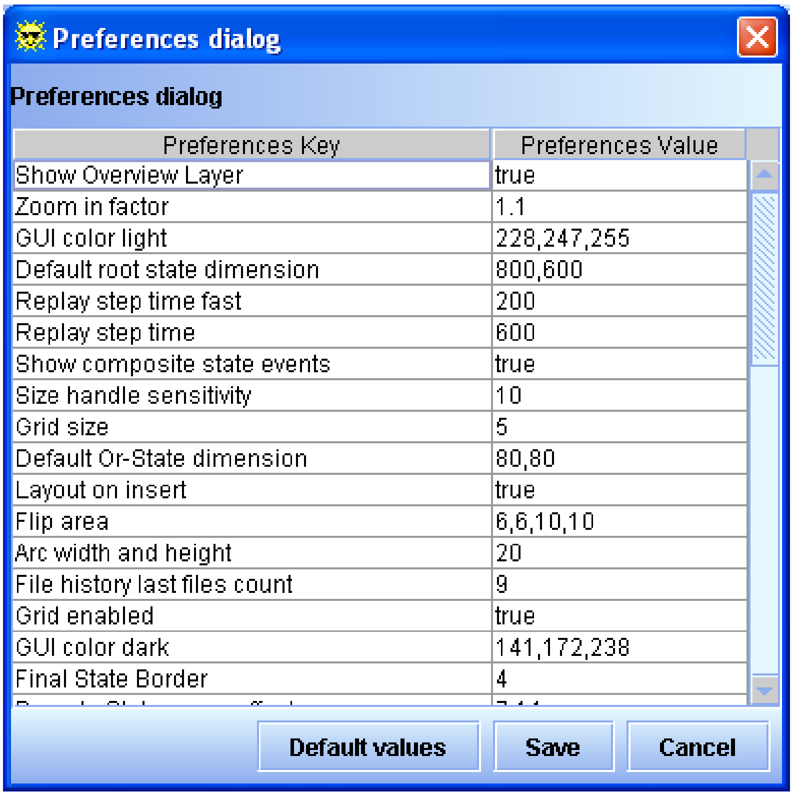


Abbildung .2.: Konfigurationsdialog

- Eine Bearbeitungsschitt-Historie, welche jede Benutzeraktion protokolliert mit Zeitstempel und Information darüber, ob die Aktion erfolgreich ausgeführt wurde. Eine *Undo-and-Redo*-Historie, welche farblich kennzeichnet, welche Aktionen rückgängig gemacht werden können, dargestellt in hellblau, und welche wiederholt werden können,dargestellt in dunkelblau. Die Historien sind exportierbar, siehe Abbildung .4, aufrufbar über den Menüeintrag *Replay→Trace Log*.

- Zoom, gleitend über den Regler in der Statusleiste und in konfigurierbaren Schritten über das Menü *Format*

- Sprache wahlweise Englisch und Deutsch, einstellbar über den Menüeintrag *File→Preferences* und dort den Eintrag *Language*

- Aktuelle Maus-Koordinaten-Anzeige und *Undo-and-Redo*-Schritt-Anzeige in der Statusleiste

- Werkzeugleisten-Gruppen wahlweise einblendbar über den Menüeintrag *File-Preferences* oder im Kontextmenü, welches über der freien Fläche der Werk-

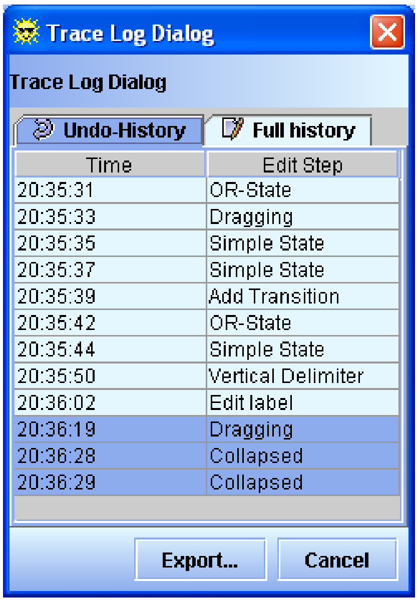


Abbildung .3.: Unterdrückbare Fehlermeldung

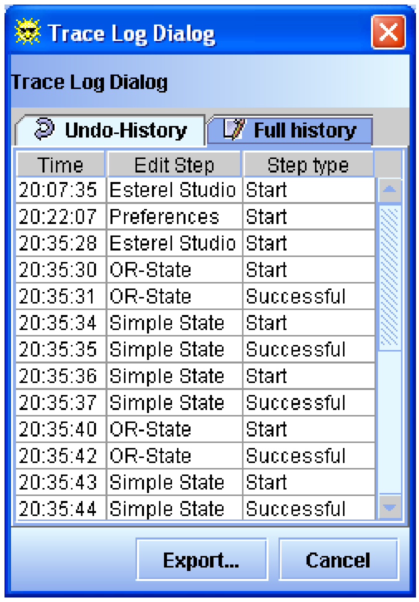
zeugleiste geöffnet wird.

.1.2. Funktionen zur Bearbeitung eines Statecharts

- Es gelten die in Abschnitt 4.1 angeführten Regeln.
- Morphing bei allen automatischen Plazierungsmechanismen und Layouts. Die Geschwindigkeit ist konfigurierbar.
- Kollabieren und Expandieren von *OR-States* und *AND-States*. Beim Kollabieren werden die innereren Zustände unsichtbar.
- Rasterorientiertes Bearbeiten mit konfigurierbarer Rasterweite
- Farbliche Syntaxprüfung: Hervorhebung unvollständiger Zustände
- Typmorphing für Diagrammelemente: Nachträgliches Ändern des Typs eines Diagrammelementes (*Simple-State* nach *OR-State*, *OR-State* nach *AND-State*, Transitionsarten beliebig, und andere. Aufrufbar ist dies über das Kontextmenü auf einem Diagrammelement oder über das Menü *Edit*.
- Für jeden Zustand kann über das Kontextmenü beliebig das *Final*-Attribut gesetzt oder gelöscht werden, welches besagt, dass dieser Zustand ein finaler oder nichtfinaler Zustand ist.
- Transitionsprioritäten werden automatisch vergeben. Bei manueller Änderung einer Transitionspriorität findet eine automatische Prioritätenanpassung aller Transitionen des gleichen Quellzustands statt.
- *Cut-and-Paste*
- *Drag-and-Drop*



(a) Undo-/ Redo-Historie



(b) Komplette Bearbeitungsschritt-Historie

Abbildung .4.: Anzeige der Bearbeitungsschritte

- Live-Preview
- Anwendung eines *KIEL*-Layouts auf das gesamte *Statechart* oder auf einzelne States, aufrufbar über das Kontextmenü.
- Layout on the fly: Automatisches *Platz schaffen* mit Morphing beim Einfügen eines neuen Zustands:
  - Wird innerhalb eines *Composite-States* ein neuer Zustand platziert, wird der *Composite-State* automatisch vergrößert, sofern nach der Vergrößerung der neue Zustand platziert werden kann. Dies ist jedoch noch nicht der Fall, wenn der neue Zustand aufgrund der Lage zukünftiger Geschwister (vorhandener Zustände) nicht platziert werden kann.
  - Ein *OR-State* und ein *AND-State* kann jederzeit ausgeklappt werden. Ist für das Ausklappen nicht genügend Platz vorhanden, so werden rekursiv alle kollidierenden Geschwister verschoben, sowie, falls notwendig, der Vater des ausklappenden Zustands vergrößert. Vergrößert sich dieser Vater, so wird der gleiche Mechanismus *eine Stufe höher* (rekursiv) angewandt.
- Editieren von *On-Entry-Actions*, *On-Inside-Actions* und *On-Exit-Actions* bei allen Zuständen, aufrufbar über das Kontextmenü. Diese werden im Inneren des jeweiligen Zustands angezeigt.

- Deklaration von lokalen Ereignissen und lokalen Variablen bei *Composite-States*, editierbar über das Kontextmenü. Diese werden unter dem Zustandsnamen im Inneren des Zustands angezeigt.

- Editieren von *statechart*-globalen *Input-Events* und *Output-Events*. Diese werden nur im Bearbeitungsdialog angezeigt, aufrufbar über den Menüeintrag *Edit→Edit Statechart input events* und *Edit→Edit Statechart output events*.

- Editieren von Transitionsbeschriftungen

- Relatives Ausrichten von States an anderen States und dasTrimmen auf eine gemeinsame Größe: Es müssen mehrere Zustände mittels *CTRL*-Taste selektiert werden, wobei am ersten Element ausgerichtet wird.

- Verschiedene Kurvenarten für Transitionen: Spline, Bezier, orthogonal, wählbar über das Kontextmenü

- Nachträgliches Ändern der Hierarchie: Kind-Zustände können herausgezogen werden.

.2. Benutzungshinweise für Editorfunktionen

Bei der ersten Arbeit mit dem Editor ist es empfehlenswert, die Standardwerte für Benutzereinstellungen zu aktivieren, welche später über den Menüeintrag *File→Preferences* geändert werden können. Die Aktivierung der Standardeinstellungen wird erreicht durch *File→Preferences* und im sich öffnenden Dialog mit dem Button *Default Values*.

Im nachfolgenden werden Benutzungshinweise für Editorfunktionen, welche nicht intuitiv ausführbar sind, gegeben.

Einstellung der Morphing-Geschwindigkeit :

Die Art der Einstellung der Geschwindigkeit ist abhängig vom Betriebssystem. Grundsätzlich auf jeder Plattform kann mittels der Benutzereinstellung *morphingLayouterStepInterval* die Geschwindigkeit eingestellt werden. Sinnvolle Werte liegen bei 50-300. Formel: *morphingLayouterStepInterval* / Anzahl Diagrammelemente = Maximalpixeländerung je Graphik-Refresh-Zyklus. Für Windows-Rechner ist auch die direkte Angabe der Geschwindigkeit in Pixels je Sekunde möglich mit der Benutzereinstellung *morphingLayouterSpeed*. Sinnvolle Werte liegen hier um 200.

Nur wenn *morphingLayouterStepInterval* auf den Wert Null gesetzt ist, wird der Wert von *morphingLayouterSpeed* berücksichtigt. Die passenden Werte sind abhängig von der zur Verfügung stehenden Rechenleistung.

Selektion von Diagrammelementen :
Bei der Selektion von Diagrammelementen mittels CTRL-Taste – dies wird bei Mehrfachselektion für relatives Ausrichten und Verschieben von mehreren Elementen – ist darauf zu achten, dass das zu selektierende Element mehrmals geklickt werden muss, bis es tatsächlich selektiert ist. Welches Element nach einem Mausklick selektiert ist, ist erkennbar an den *Size-Handles*.

Entfernen eines Elements aus einer Region :
Durch einen bisher nicht behobenen Softwarefehler ist es notwendig, nach dem Entfernen des letzten Kind-Elements einer Region, sei es durch Herausziehen oder direktes Löschen, den umgebenden *AND-State* einmal einzuklappen und anschließend wieder aufzuklappen.

Preferences-Werte ändern : Nach Änderung eines Wertes in der Tabelle der Benutzereinstellungen muss die Eingabe mit der Entertaste abgeschlossen werden.

Erzeugen von Zuständen :
Bei der Erzeugung von Zuständen ist zu beachten, dass vor jeder Hinzufüge-Operation der Editor in den Hinzufüge-Modus geschaltet werden muss, indem das gewünschte Element angewählt wird in der Werkzeugleiste oder im Menü *Insert*. Nach dem Platzieren des Elements ist der Editor wieder im Selektionsmodus. Beim Platzieren ist auf den Mauszeiger zu achten: Erscheint ein rotes Kreuz, dann ist die aktuelle Mausposition für das Erzeugen des Elementes nicht erlaubt. Hierbei ist zu bedenken, dass ein Mindestabstand zu anderen Elementen gewahrt bleiben muß und dass an bestimmten Orten das gewünschte Element nicht zulässig ist.

Erzeugen von Transitionen :
Zunächst muß der Mauszeiger genau in der Mitte des Quellzustands platziert werden, so dass er sich in einen Hand-Mauszeiger verwandelt. Bei Mausklick wird der selektierte Quellzustand umrahmt. Dann wird der aktuelle Punkt geklickt und die Transition gezogen, indem der Mauszeiger über der Mitte des Zielzustands so platziert wird, dass dieser Zustand optisch durch einen Rahmen hervorgehoben wird. Dann ist das Ziehen zu beenden. Der Transitionstyp wird durch den Quellzustand bestimmt, siehe Abschnitt 5.3.3, und durch die Auswahlbox in der Werkzeugleiste.

Transitionen ändern :
Ein Stützpunkt ist hinzufügbar, indem an der gewünschten Stelle auf der Transition mit Shift+Kontextmenü-Maustaste geklickt wird. Dieser Stützpunkt kann verschoben werden, indem der Mauszeiger derart auf dem Stützpunkt platziert wird, dass sich der Mauszeiger in ein Kreuz verwandelt. Dann kann gezogen werden.

Quellzustand und Zielzustand können geändert werden, indem der Mauszeiger so auf eine der Transitionsenden platziert wird, dass sich der Mauszeiger

in ein Kreuz verwandelt. Dann kann dieser Punkt verschoben werden, wobei darauf zu achten ist, dass der Zielpunkt sich in der Mitte eines anderen Zustands befindet, genau wie beim Zeichnen einer neuen Transition.

Selektieren und Verschieben von kleinen Diagrammelementen :
Für das Selektieren und Verschieben von kleinen Elementen wie z. B. *Pseudo-States* empfiehlt es sich, das Erzeugen von Transitionen temporär auszuschalten durch Klick auf den Menüeintrag *Insert→Transition Mode*.

Das Ziehen des Mauszeigers auf einem Diagrammelement wird je nach Position des Mauszeigers vom Editor unterschiedlich interpretiert: Befindet sich der Mauszeiger in der Mitte eines Zustands, so wird erwartet, dass der Benutzer eine Transition erzeugen möchte. Sofern sich der Mauszeiger allerdings am Rand eines Zustands befindet, so wird angenommen, dass die Größe des Zustands geändert werden soll. Befindet sich der Mauszeiger an einer anderen Stelle innerhalb des Zustands, so wird ein Ziehen des Mauszeigers den Zustand verschieben. Bei kleinen Diagrammelementen jedoch ist es kaum möglich, eine Stelle innerhalb des Zustands zu finden, welche nicht das Erzeugen einer Transition oder eine Größenänderung sondern das Verschieben des Zustands bewirkt.

Relatives Ausrichten :
Ausgerichtet wird immer an dem ersten von mehreren selektierten Zuständen. Mehrere Zustände werden selektiert mit Hilfe der *CTRL*-Taste. Der Mauszeiger ist über den ersten zu selektierenden Zustand zu positionieren, dann ist die *CTRL*-Taste zu drücken und gedrückt zu halten, dann mehrmals der Zustand anzuklicken, bis der gewünschte Zustand selektiert ist. Danach ist die *CTRL*-Taste gedrückt zu halten und weitere Zustände auf gleiche Weise der Selektion hinzuzufügen. Hiernach kann eine Ausrichtungsfunktion im Menü oder der Werkzeugleiste ausgewählt werden.

Bearbeiten der Variablendeklaration eines Zustands :
Über das Kontextmenü, Eintrag *Variables*, kann die Deklaration von Variablen für den betreffenden Zustand vorgenommen werden. Die bereits deklarierten Variablen werden mit ihrem Typ kommasepariert im Dialogfenster angezeigt. Es können Variablen gelöscht und hinzugefügt werden. Eine vollständige Variablendeklaration hat die Form *varname : typname*. Die erlaubte verkürzte Schreibweise *varname*, also ohne Typangabe, bedeutet das gleiche wie *varname : integer*. Fehlt die Angabe des Typs einer Variablen, so ist ihr Typ also *integer*.

Ausblenden von Benutzeroberflächen-Elementen :
Mit Mausklick auf die freie Fläche der Werkzeugleiste erscheint ein Kontext-Menü, in welchem die Sichtbarkeit von Werkzeugelementen sowie der Statusleiste eingestellt werden kann. Soll diese Einstellung dauerhaft geändert werden, so ist diese Änderung unter *File→Preferences* vorzunehmen.

Benutzereinstellungen :
Siehe hierzu .2.1

.2.1. Einstellungen

Über den Menüeintrag *File→Preferences* lassen sich Anwendungseigenschaften einstellen. Änderungen werden sofort aktiv. Sollen die Änderungen von Dauer sein, so muss nach dem Ändern auf *Speichern* geklickt werden. Ein Klick auf *Abbrechen* stellt nicht die Ursprungswerte wieder her. Im Folgenden wird für jede Eigenschaft die Wertemenge angegeben. *bool* steht für die Zeichenkette *true* oder *false*, *int* steht für eine natürliche Zahl, *float* steht für eine Fließkommazahl, *string* steht für eine Zeichenkette.

Show overview layer :
Wertemenge [bool];
Soll die Übersichtsdarstellung angezeigt werden?

Zoom in factor :
Wertemenge [float]; sinnvoll sind Werte > 1.
Um wieviel soll vergrößert werden bei Klick auf den *Zoom* +-Button?

Zoom out factor :
Wertemenge [float]; sinnvoll sind Werte < 1.
Um wieviel soll verkleinert werden bei Klick auf den *Zoom* --Button?

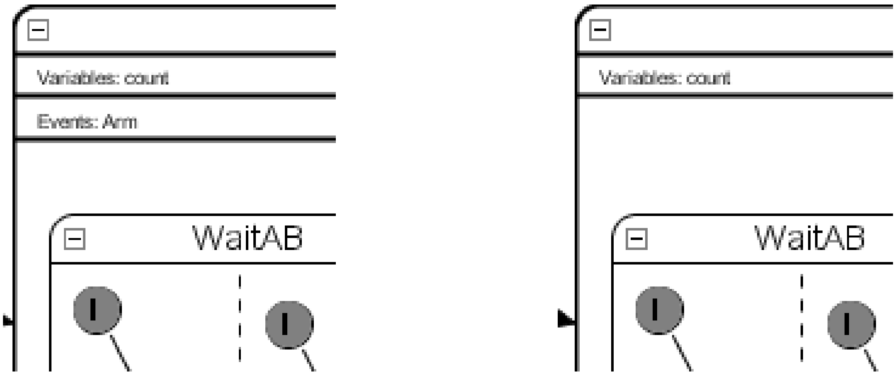
Trace Log separator :
Wertemenge [string]; sinnvoll sind Zeichenketten wie ';' oder ','.
Welcher Separator soll verwendet werden bei Export von *Tracelog*-Daten? Siehe hierzu Abbildung 4.4.

Default root state dimension : Wertemenge [int, int]; sinnvoll sind Werte, welche zur Monitorauflösung passen.
Definiert die initiale Größe des Anwendungsfensters

Show composite State events :
Wertemenge [bool];
Sollen die *Events* eines *Composite-States* angezeigt werden? Siehe Abbildung .5. Dies kann auch als temporäre Einstellung über das Menü gesteuert werden.

Show composite state variables :
Wertemenge [bool];
Sollen die Variablen eines *Composite-States* angezeigt werden? Siehe Abbildung .5. Dies kann auch als temporäre Einstellung über das Menü gesteuert werden.

Show state actions :
Wertemenge [bool];
Sollen Zustandsaktionen angezeigt werden?



(a) *Events* werden angezeigt (b) Events werden ausgeblendet

Abbildung .5.: Anzeige der *Events*

Replay step time fast :
Wertemenge [int]; sinnvoll sind Werte 0 - 10000 (Millisekunden).
Wie groß soll der zeitliche Abstand zwischen zwei Schritten sein beim schnellen Abspielen von Bearbeitungsschritten? Siehe hierzu Abschnitt 4.9.

Replay step time : Wertemenge [int]; sinnvoll sind Werte 0 - 10000 (Millisekunden).
Wie groß soll der zeitliche Abstand zwischen zwei Schritten sein beim normalen Abspielen von Bearbeitungsschritten? Siehe hierzu Abschnitt 4.9.

Show alignment toolbar on load :
Wertemenge [bool];
Sollen die Ausrichtungs-Buttons initial in der Werkzeugleiste angezeigt werden? Dies kann auch als temporäre Einstellung im Kontextmenü zum Hauptmenü eingestellt werden, siehe hierzu Abschnitt 4.19.

Show replay toolbar on load : Wertemenge [bool];
Soll die *Replay*-Werkzeugleiste initial angezeigt werden? Dies kann auch als temporäre Einstellung im Kontextmenü zum Hauptmenü eingestellt werden, siehe hierzu Abschnitt 4.19.

Grid size :
Wertemenge [int]; sinnvoll sind Werte 1 - 200 und der Wert muß kleiner sein als der der 'Intersection sensitivity'
Wie groß soll das Raster sein, welches angezeigt werden kann und welches bei dem rasterorientierten Bearbeiten eine Rolle spielt? Siehe Abschnitt 4.8.

Intersection sensitivity :

Wertemenge [int]; sinnvoll sind Werte von 0 - 20 und der Wert muß größer sein als der der *Grid Size*.
Ab welcher Entfernung zweier Zustände sollen die beiden als sich schneidend interpretiert werden?

Grid enabled :
Wertemenge [bool];
Soll rasterorientiertes Bearbeiten eingeschaltet sein? Siehe Abschnitt 4.8.

Grid visible :
Wertemenge [bool];
Soll das Bearbeitungsraster sichtbar sein? Davon unabhängig ist, ob rasterorientiertes Bearbeiten eingeschaltet sein soll, siehe Abschnitt 4.8 und insbesondere Abbildung 4.3.

Default OR-State dimension :
Wertemenge [int, int]; sinnvoll sind Werte um 100.
Wie groß sollen *OR-States* initial sein?

Default Node dimension :
Wertemenge [int, int];
Wie groß sollen *Pseudo-States* initial sein?

Default state dimension :
Wertemenge [int ,int]
Wie groß sollen *Simple-States* initial sein?

Layout on insert :
Wertemenge [bool];
Soll beim Absetzen eines neu zu erzeugenden Zustands automatisch Platz geschaffen werden? Der Prozess des Absetzens dauert so lange, bis die Maustaste losgelassen wird, siehe Abschnitt 4.13.

Flip Area :
Wertemenge [int, int, int, int]; sinnvoll sind Werte 1 - 30.
In welchem Bereich eines *OR-States* und *AND-States* soll bei Mausklick in diesen Bereich dieser Zustand kollabieren oder expandieren? Siehe hierzu Abschnitt 4.5. Die vier Werte beschreiben ein Rechteck mit x-Koordinate, y-Koordinate, Breite und Höhe.

Arc width and height :
Wertemenge [int]; sinnvoll sind Werte 0 - 40.
Wie weit soll die Rundung der Ecken von Zuständen sein?

File history last files count :
Wertemenge [int]; sinnvoll sind Werte um 10.
Wie viele Dateien sollen im Menü *File→Last Files* angezeigt werden? Unabhängig von dieser Einstellung erhalten nur die ersten neun Einträge zusätzlich einen *Key-Shortcut*.

Final State border :
Wertemenge [int]; sinnvoll sind Werte 1 - 10.
Wie groß soll der Abstand der beiden Rahmenlinien von *Final-States* sein?

Pseudo State name offset :
Wertemenge [int, int]; sinnvoll sind Werte von 0 - 20.
Wo soll der den *Pseudo-State* beschreibende Buchstabe ('I', 'S', 'C') innerhalb des *Pseudo-States* gezeichnet werden? Eine Änderung des Standardwertes kommt nur in Betracht, wenn alle *Pseudo-States* eine neue Standardgröße erhalten sollen.

Language :
Wertemenge [string]; Erlaubt sind 'en' und 'de'.
Setzt für alle sprachlichen Anzeigeelemente des Editors (Dialoge, Menüeinträge) die Sprache auf deutsch ('de') oder Englisch ('en'). Siehe hierzu Abbildung 4.12.

Line end diameter :
Wertemenge [int]; sinnvoll sind Werte um 6.
Wie groß soll der grüne und rote Kreis am Ende einer *Strong-Abortion* und *Normal-Termination* sein?

Label offset :
Wertemenge [int, int]; sinnvoll sind Werte um 0.
Wo soll die Transitionsbeschriftung positioniert werden?

Initial root state offset :
Wertemenge [int]; sinnvoll sind Werte > 0.
Mit welchem Offset soll das *Statechart* gezeichnet werden auf der Zeichenfläche?

MorphingLayouter check interval :
Wertemenge [int]; sinnvoll sind Werte um 10.
Nach wie vielen Iterationsschritten soll der Geschwindigkeitsregelmechanismus des Morphing die Geschwindigkeit überprüfen?

Cell selection tolerance :
Wertemenge [int]; sinnvoll sind Werte um 10.
Ein Diagrammelement wird selektiert, sobald der Mauszeiger über diesem Element bewegt wird. Allerdings kann ein Diagrammelement auch selektiert werden, indem es angeklickt wird. Wie nahe muss der Mauszeiger bei Mausklick bei einem Diagrammelement sein, damit dieses selektiert wird?

Place of Deep-History-State :
Wertemenge [int, int]; sinnvoll sind Werte > 0.
Wo soll ein *Deep-History-State* platziert werden innerhalb eines *Composite-States*?

Composite State image offset :
Wertemenge [int]; sinnvoll sind Werte um 5.
Wo soll das Plus- und Minus-Zeichen für das Kollabieren und Expandieren eines *Composite-States* platziert werden?

Morphing layouter speed :
Wertemenge [int]; sinnvoll sind Werte um 200.
Wie schnell soll ein Morphing durchgeführt werden? Angabe in Pixeln pro Sekunde.

border offset for valid area of composite states :
Wertemenge [int]; sinnvoll sind Werte um 10.
Wie nahe dürfen Kind-Elemente am Rahmen ihres Vaters liegen?

Show splash sceen :
Wertemenge [bool];
Soll das Startbild angezeigt werden beim Laden der Anwendung?

Zoom out factor :
Wertemenge [float]; sinnvoll sind Werte < 1.

Morphing layouter border distance :
Wertemenge [int]; sinnvoll sind Werte um 50.
Wie nahe dürfen aufklappende Zustände an andere Zustände herankommen und von welchem Abstand an diese anderen Zustände verschoben werden?

Show Collapse Buttons :
Wertemenge [bool];
Sollen die Plus- und Minus-Zeichen bei *OR-States* und *AND-States* angezeigt werden? Auch wenn die Zeichen nicht angezeigt werden, können diese *Composite-States* kollabiert und expandiert werden.

Overview layer color :
Wertemenge [int, int, int]; Erlaubt sind nur Werte <= 255.
Welche Farbe soll die Übersichtsdarstellung haben?

GUI color light :
Wertemenge [int,int,int]; Erlaubt sind nur Werte <= 255.
Die Anwendung kennt zwei Hauptfarben: im Standardfall sind dies hellblau und dunkelblau. Mit dieser Eigenschaft lässt sich die hellblaue Farbe ändern.

GUI color dark :
Wertemenge [int,int,int]; Erlaubt sind nur Werte <= 255.
Die Anwendung kennt zwei Hauptfarben: im Standardfall sind dies hellblau und dunkelblau. Mit dieser Eigenschaft lässt sich die dunkelblaue Farbe ändern.

GUI selected button light :
Wertemenge [int, int, int]; Erlaubt sind nur Werte <= 255.
Welche helle Farbe soll der *Toggle-Button* in der Werkzeugleiste haben?

GUI selected button dark :
Wertemenge [int, int, int]; Erlaubt sind nur Werte <= 255.
Welche dunkle Farbe soll der *Toggle-Button* in der Werkzeugleiste haben?

A. Java-Programm-Quelltext

In diesem Teil des Anhangs wird im Abschnitt A.1 ein Überblick über den Programm-Quelltext gegeben. Hieran anschließend ist der vollständige Programm-Quelltext des *KIEL*-Editors und des *KIEL*-Anwendungsfensters in Abschnitt A.4 aufgeführt.

A.1. Überblick

Zunächst wird ein nach *Packages* gegliederter Überblick über sämtliche Klassen gegeben, welche im Rahmen dieser Arbeit implementiert wurden und zusammen den *KIEL*-Editor und das *KIEL*-Anwendungsfenster bilden. Danach wird ergänzend zu Kapitel 5 im Abschnitt A.2 das Zusammenspiel der Klassen untereinander erläutert. Abschließend werden in Abschnitt A.3 Erweiterungsmöglichkeiten und häufig auftretende Ergänzungs- und Anpassungsaufgaben beschrieben.

A.1.1. *Package* `kiel.editor`

Dieses *Package* enthält gemäß dem *MVC*-Entwurfsmuster die *Model*- und *View*-Komponente des Editors. Das *Model* besteht aus folgenden Klassen:

- `MyGraphModel`: Enthält das das *Statechart* in Form eines *JGraph*- und *KIEL*-Datenmodells und auf diesen Modellen operierende Methoden.

- `MyGraphModelUtilities`: Enthält häufig verwendete Operationen auf Datenmodell-Objekten. Diese Operationen beziehen sich nicht auf ein konkretes Datenmodell und gehören daher nicht in die Klasse `MyGraphModel`. Diese Operationen werden vom `MyGraphModel` und von einigen *Controller*-Klassen verwendet.

- `UndoClusterManager`: Verwaltet Benutzer-Bearbeitungsschritte in Form von gruppierten Datenmodell-Änderungsschritten.

- `UndoClusterManagerListener`: Dies ist ein Interface, welches nur von der Klasse `Editor` implementiert wird.

- `EditStep`: Definiert die Datenstruktur, auf der der `UndoClusterManager` operiert.

- `FileHistory`: Beinhaltet eine Liste der zuletzt göffneten Dateien. Diese Komponente wird von der *KIEL*-Hauptklasse `kiel.KielFrame` verwendet.

- `EditListener`: Dieses Interface wird ausschließlich implementiert von der Klasse `ActionEditEdit`. Dieser Listener wird von der `MyBasicGraphUI`-Komponente (siehe unten) immer dann benachrichtigt, wenn das Beschriften eines Diagrammelements beendet wurde.
- `MorphingLayouter`: Führt auf dem Datenmodell eine gleitende Strukturveränderung durch. Für jeden Morphing-Vorgang wird diese Klasse instantiiert: Beim Kollabieren, Expandieren und Platzieren eines neuen Zustands und bei Anwendung eines Layouts des *KIEL*-Layouters.

Die *View* wird aus folgenden Klassen gebildet:

- `Editor`: Dies ist die Wurzelklasse des *KIEL*-Editors, welche vom Anwendungsfenster `kiel.KielFrame` instantiiert wird.
- `MyJGraph`: Diese Klasse erweitert die Klasse `org.jgraph.JGraph` und stellt die Zeichenfläche des Editors dar.
- `MyBasicGraphUI`: Bestimmt unter anderem die Interpretation von Maus-Ereignissen und passt als Subklasse von `org.jgraph.BasicGraphUI` das Verhalten des Rahmenwerks hinsichtlich des Zeichnens von hierarchischen Zuständen und der Zeichenqualität an die Anforderungen von *KIEL* an. Des Weiteren wird in dieser Klasse die Änderung des Editor-Modus initiiert.
- `PreferencesDialog`: Zeigt in einem separaten Fenster die änderbaren Konfigurationseinstellungen.
- `TraceLogDialog`: Stellt tabellarisch in einem separaten Fenster die Bearbeitungshistorie dar.
- `Utils`: Stellt dem Editor und dem Anwendungsfenster GUI-Komponenten zur Verfügung, welche ein einheitliches Aussehen haben, ihre Textelemente bei Änderung der Sprache sofort ändern und welche zum Teil per *Drag-and-Drop* in der Benutzeroberfläche verschoben werden können.
- `MyDefaultCellViewFactory`: Erzeugt die Objekte, welche in einer Instanz der Klasse `MyJGraph` – der *View* – die Diagrammelemente repräsentieren.

A.1.2. *Package* `kiel.editor.controller`

Dieses *Package* enthält den *Controller* des *KIEL*-Editors hinsichtlich des *Model-View-Controller*-Entwurfsmusters. Dies sind allesamt `EditorAction`-Klassen, einige von ihnen sind darüber hinaus auch `EditorMode`-Klassen. Insgesamt sind in diesem *Package* 78 Klassen zusammengefaßt.

- `ActionXY`: XY steht für einen Menüeintrag, z. B. ist `ActionEditRedo` eine Operation, welche ausgeführt wird, wenn der Menüeintrag *Edit*→*Redo* gewählt wird.

- `EditorAction`: Dies ist die Basisklasse für alle `ActionXY`-Klassen.

- `EditorActions`: Von jeder `ActionXY`-Klasse wird nur eine Instanz erzeugt. Diese Instanzen werden in der Klasse `EditorActions` gehalten.

- `EditorModeXY`: Hier steht XY für einen Editor-Modus, der durch Klick auf einen Diagrammelement-Button in der Werkzeugleiste gewählt wird. `EditorModeAddOrState` beschreibt z. B. das Verhalten des Editors, nachdem der Benutzer den Button für das Hinzufügen eines *OR-States* geklickt hat. Eine Ausnahme bildet die Klasse `EditorModeSelectOrDragOrResizeOrFlipExpand`: Wie der Name besagt, repräsentiert diese Klasse den Editor-Modus, der durch Klick auf den Selektionspfeil-Button in der Werkzeugleiste gewählt wird.

- `EditorMode`: Dies ist die Basisklasse für alle `EditorModeXY`-Klassen und ist selbst abgeleitet von `EditorAction`.

- `EditorModes`: Von jeder `EditorModeXY`-Klasse wird nur eine Instanz erzeugt. Diese Instanzen werden in der Klasse `EditorModes` gehalten. Des Weiteren wird hier der aktuelle Editor-Modus als Instanz einer `EditorMode`-Klasse gesetzt.

- `MyMarqueeHandler`: Diese Klasse repräsentiert das Verhalten des Editors beim Erzeugen einer Transition. Die Namensgebung ist durch die *JGraph*-Struktur bedingt.

A.1.3. *Package* `kiel.editor.graph`

Dieses *Package* enthält die *KIEL*-Editor-Erweiterungen der `CellView`- und `GraphCell`-Implementationen von *JGraph* zur Darstellung unterschiedlicher Diagrammelement-Formen.

- `XYCell`: Für jeden Diagrammelement-Typ XY aus der *KIEL*-Datenstruktur wird hier eine Klasse bereitgestellt, für die Klasse `kiel.datastructure.ORState` z. B. ist dies die Klasse `kiel.editor.graph.ORStateCell`. Hierbei wird die Klassenhierarchie aus der *KIEL*-Datenstruktur übernommen. `XYCell`-Instanzen bilden das Datenmodell, auf dem das Editor-*Model* operiert.

- `XYView`: Zu jeder `XYCell`-Klasse gibt es eine `XYView`-Klasse. `XYView`-Instanzen bilden das Datenmodell, auf dem die Editor-*View* operiert.

- `XYRenderer`: Zu Diagrammelementen, welche sich in ihrer graphischen Repräsentation von anderen Typen unterscheiden, wird eine `XYRenderer`-Klasse bereitgestellt. Diese Klassen beinhalten in Form von Graphik-Primitiven die Beschreibung der graphischen Repräsentation eines Diagrammtyps.

- `CellViewRendererFactory`: Von jeder `XYRenderer`-Klasse wird nur eine Instanz erzeugt. Diese Instanzen werden in der Klasse `CellViewRendererFactory` gehalten.
- `MyGraphConstants`: Jedes `XYCell`-Objekt besitzt eine Menge von Attributen, welche durch die Klassen `org.jgraph.GraphConstants` und `MyGraphConstants` gesetzt werden. Hierbei ist die Klasse `MyGraphConstants` zuständig für diejenigen Attribute, welche durch die Implementierung des Editors hinzukommen.

A.1.4. *Package* `kiel.editor.resources`

Dieses *Package* enthält Klassen für den Zugriff auf Anwendungsressourcen wie Bilder, sprachabhängige Texte und Konfigurationseinstellungen.

- `Preferences`: Lädt und speichert Konfigurationseinstellungen, welche über den `PreferencesDialog` vom Benutzer geändert werden.
- `ResourceBundle`: Stellt einen Zugriff auf spracheinstellungsabhängige Texte bereit, welche als Beschriftungen von Menüeinträgen und *Tooltips* dienen.
- `ResourceLoader`: Lädt Bilder als `ImageIcon`-Objekte.
- `MyPreferences.properties`: Enthält für jede Konfigurationseinstellung einen Standardwert.
- `MyResources.properties`: Enthält die *Key-Shortcuts* für alle Menüeinträge.
- `MyResources_de.properties`: Enthält alle deutschsprachigen Texte für Beschriftungen und Meldungen der Benutzeroberfläche.
- `MyResources_en.properties`: Enthält alle englischsprachigen Texte für Beschriftungen und Meldungen der Benutzeroberfläche.

A.2. Zusammenspiel der Klassen

A.2.1. Kommunikation zwischen *View* und *Controller*

Die *Controller*-Klassen werden als *Singletons* [14] in einem *Cache* in den Klassen `EditorActions` und `EditorModes` verwaltet. Beim Aufbau der Menüs und der Werkzeugleiste, welcher in der Klasse `Editor` erfolgt, werden die *Controller*-Klassen instantiiert und als `java.awt.ActionListener` mit den Menüelementen und Werkzeugleisten-Buttons assoziiert. Jeder *Controller* ist abgeleitet von der Klasse `EditorAction`.

Bei Instantiierung der Klasse `Editor` durch das `KielFrame` wird die `Editor`-Instanz in einer statischen Variable der Klasse `EditorAction` gespeichert. Damit erhält jede *Controller*-Klasse die Möglichkeit, auf das `Editor`-Objekt und damit

auf die gesamte *View* des *KIEL*-Editors zuzugreifen. Diese Richtung der Kommunikation ist im *MVC*-Entwurfsmuster nicht vorgesehen. Allerdings verstößt dies nicht gegen das Entwurfsmuster, da im *KIEL*-Editor in dieser Richtung lediglich Eigenschaften der *View* *abgefragt* werden wie z. B. diejenige des aktuell selektierten Diagrammelements. Eine Manipulation der *View* hingegen findet nicht statt. Des Weiteren erhält ein *Controller* mittels dieser Referenz auf das `Editor`-Objekt Zugriff auf das *Model* in Form von `getEditor().getGraph().getMyGraphModel()` und `getEditor().getGraph().getUndoClusterManager()`.

A.2.2. Kommunikation von *Controller* nach *Model*

Der *Controller* ruft über `getEditor().getGraph().getMyGraphModel()` Methoden des *Models* auf, welche das Datenmodell ändern. In der Regel ist dies genau *ein* Methodenaufruf. Generell wird jeder *Controller*, welcher eine Änderung auf dem *Model* durchführt, eine Transaktionsklammer setzen, um zum einen einen Eintrag in der Bearbeitungshistorie zu erzeugen und zum anderen einen Bearbeitungsschritt zu definieren für die *Undo-and-Redo*-Funktionen.

```
MyJGraph graph = getEditor().getGraph();

graph.getUndoClusterManager().setUndoClusterStart();
DefaultGraphCell cell = graph.getSelectionCell();
...
graph.getMyGraphModel().addTransition(...);
graph.getUndoClusterManager().setUndoClusterEnd(this);
```

Abbildung A.1.: Ausschnitt aus einer *Controller*-Klasse

Abbildung A.1 zeigt einen Programm-Quelltext-Ausschnitt aus der Klasse `Action-AddLoopTransition`.

A.2.3. Struktur der *View*-Klassen

Die Klasse `Editor` ist keine *Java-Swing*-Komponente sondern vom Typ `kiel.util-.main.KielComponent`. Sie beinhaltet alle Benutzeroberfächen-Objekte des *KIEL*-Editors, indem sie ein `MyJGraph`-Objekt (siehe Abschnitt A.1.1, die Menüleiste, die Werkzeugleiste mitsamt ihren Elementen und eine mehrschichtige *Java-Swing*-Komponente referenziert, welche zwei `MyJGraph`-Objekte auf je einer Schicht anordnet. Eines der beiden `MyJGraph`-Objekte – die Bearbeitungsschicht – enthält eine Referenz auf das weitere `MyJGraph`-Objekt – die Überischtsschicht. Wann immer sich etwas an dem Bearbeitungsschicht-`MyJGraph`-Objekt ändert, benachrichtigt es sein referenziertes Übersichtsschicht-`MyJGraph`-Objekt.

A.2.4. Der `UndoClusterManager`

Diese Klasse ist ein *Wrapper* für ein `org.jgraph.graph.GraphUndoManager`-Objekt, seinerseits abgeleitet von `javax.swing.undo.UndoManager`. Wann immer am `MyJGraph`-Objekt eine neue `MyGraphModel`-Instanz als *Model* gesetzt wird, wird in dem Wrapper ein neues `org.jgraph.graph.GraphUndoManager`-Objekt erzeugt und dieses als `UndoableEditListener` am neuen *Model* registriert.

Der `Editor` ist der einzige Beobachter des `UndoClusterManagers`. Wann immer ein *Undo*- oder `Redo`-Schritt durchgeführt wird, sich also einer der Stacks des `UndoClusterManager` ändert (siehe Abschnitt 5.3.8), wird der `Editor` benachrichtigt. Dieser aktualisiert dann die Anzeige der verfügbaren *Undo-and-Redo*-Schritte in der Statusleiste des *KIEL*-Editors (siehe Abschnitt 4.9).

A.3. Erweiterungs- und Anpassungsmöglichkeiten

In diesem Abschnitt wird nur ein kurzer Auszug aus der Umsetzung möglicher Änderungsanforderungen gegeben. Die im folgenden beschriebenen Anpassungsmöglichkeiten können ohne Programmierkenntnisse umgesetzt werden.

A.3.1. Änderung eines Editor-Menüeintrags

Soll der angezeigte Name geändert werden, so ist in der Datei `MyResources_en.properties` und `MyResources_de.properties` der Text entsprechend zu ändern. Diese Datei enthält Schlüssel-Wert-Paare. Zu finden ist der zu ändernde Text unter dem Schlüssel, welcher den Menüeintrag beschreibt. *ActionFilePreferences* bezeichnet z. B. den Eintrag *File→Preferences*. Jeder Menüeintrag besitzt in der Datei `MyResources.properties` des Weiteren einen *Key-Shortcut*. Das Schlüssel-Wert-Paar `acceleratorForActionFormatAlignLeft=2F5` in dieser Datei bedeutet z. B., dass für den Menüeintrag *Format→Align-Left* der *Key-Shortcut 2F5* gelten soll. *F5* steht für die gleichnamige Funktionstaste. An dieser Stelle kann auch ein Buchstabe oder z. B. *DEL* für die Löschtaste stehen. Die *2* steht für die *CTRL*-Taste. Weiterführende Informationen über diese Art der Codierung enthält die Dokumentation zu der Klasse `javax.swing.KeyStroke`.

Soll das angezeigte Bild zu einem Menüeintrag geändert werden, so ist die Bilddatei im *GIF*-Format im Package `kiel.editor.resources` abzulegen und der Dateiname in der entsprechenden *Controller*-Klasse `ActionXY` zu ändern.

A.3.2. Ändern der Menüstruktur im *KIEL*-Anwendungsfenster

Hauptmenü und *File*-Menü des Hauptfensters sind aufgeteilt in Bereiche, welche von der Klasse `KielFrame` bereitgestellt werden und welche, die vom *KIEL*-Browser und *KIEL*-Editor definiert werden.

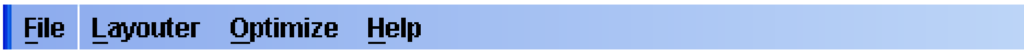
Wie in der Abbildung A.2 zu sehen, sind die variablen Bereiche diejenigen über dem *Open...*-Menüeintrag und zwischen dem *Open...*- und *Last Files*-Menüeintrag.



(a) In der Browser-Ansicht (b) In der Editor-Ansicht

Abbildung A.2.: Das *File*-Menü

Abbildung A.3 zeigt eine ähnliche Aufteilung des Hauptmenüs. Entsprechend dem Bereich, in dem eine Änderung der Menüstruktur erfolgen soll, sind die Klassen `KielFrame`, `Editor` oder `Browser` anzupassen. Die hierfür relevanten Methoden – definiert im Interface `KielComponent` – sind `getFileMenuItems1()`, `getFileMenuItems2()` und `getRootMenus()`.



(a) In der Browser-Ansicht



(b) In der Editor-Ansicht

Abbildung A.3.: Das Hauptmenü

A.3.3. Erweiterung um andere *Statechart*-Dialekte

Zur Zeit sind im Menü *File*→*New* die Einträge *Esterel Studio* und *Matlab Model* vorhanden. Das Attribut `StateChart.ModelSource` im *KIEL*-Datenmodell erhält als Wert die Menüeintrag-Beschriftung. Das Attribut `StateChart.ModelVersion` wird konstant auf den Wert *5.0* gesetzt. Die Auswahl der Werkzeugleisten-Elemente entsprechend dem gewählten *Statechart*-Dialekt im Menü *File*→*New* erfolgt in der Methode `Editor.updateToolBarAndMenuBar()`.

Die Implementierung sieht nur unterschiedliche Werkzeugleistenelemente für verschiedene *Statechart*-Dialekte vor. Andere Bereiche wie die Regeln der Syntaxprüfung und des syntaxgerichteten Editierens weisen bei jedem gewählten *Statechart*-Dialekt das gleiche Verhalten auf.

A.4. Der Quelltext

Der nachfolgende Programm-Quelltext wurde bereinigt um alle Fehler und Warnungen, welche der *Javac*-Kompiler, der *Javadoc*-Dokumentation-Generator sowie die Werkzeuge *JLint* sowie *Check Style* mit seinen *Sun-Coding-Conventions* liefern. Der vorliegende Programm-Quelltext ist daher mit einem *Java*-Kompiler ab Version 1.4 kompilierbar, sofern die vom Editor verwendeten *KIEL*-Komponenten sowie das *JGraph*-Rahmenwerk in Version 5.4 als Kompilat vorliegen.

kiel.editor.Editor

```java
package kiel.editor;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.MouseWheelEvent;
import java.awt.geom.Point2D;
import java.util.ArrayList;

import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JComboBox;
import javax.swing.JComponent;
import javax.swing.JLabel;
import javax.swing.JLayeredPane;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSlider;
import javax.swing.JTabbedPane;
import javax.swing.JToggleButton;
import javax.swing.JToolBar;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

import kiel.dataStructure.StateChart;
import kiel.editor.controller.ActionAddTransitionToggle;
import kiel.editor.controller.ActionEditCollapsed;
import kiel.editor.controller.ActionEditCopy;
import kiel.editor.controller.ActionEditCut;
import kiel.editor.controller.ActionEditEdit;
import kiel.editor.controller.ActionEditFind;
import kiel.editor.controller.ActionEditPaste;
import kiel.editor.controller.ActionEditRedo;
import kiel.editor.controller.ActionEditRemove;
import kiel.editor.controller.ActionEditSelectAll;
import kiel.editor.controller.ActionEditStateChartInputEvents;
import kiel.editor.controller.ActionEditStateChartOutputEvents;
import kiel.editor.controller.ActionEditUndo;
import kiel.editor.controller.ActionFileExport;
import kiel.editor.controller.ActionFileNewEsterelStudio;
import kiel.editor.controller.ActionFileNewMatlab;
import kiel.editor.controller.ActionFilePreferences;
import kiel.editor.controller.ActionFilePrint;
import kiel.editor.controller.ActionFileSave;
import kiel.editor.controller.ActionFileSaveAs;
import kiel.editor.controller.ActionFormatAlignBottom;
import kiel.editor.controller.ActionFormatAlignCenterHorizontal;
import kiel.editor.controller.ActionFormatAlignCenterVertical;
import kiel.editor.controller.ActionFormatAlignJustifyHorizontal;
import kiel.editor.controller.ActionFormatAlignJustifyVertical;
import kiel.editor.controller.ActionFormatAlignLeft;
import kiel.editor.controller.ActionFormatAlignRight;
import kiel.editor.controller.ActionFormatAlignTop;
import kiel.editor.controller.ActionFormatZoom100;
import kiel.editor.controller.ActionFormatZoomIn;
import kiel.editor.controller.ActionFormatZoomOut;
import kiel.editor.controller.ActionLayout;
import kiel.editor.controller.ActionLayoutSelectedState;
import kiel.editor.controller.ActionViewGrid;
import kiel.editor.controller.ActionViewOverview;
import kiel.editor.controller.ActionViewReplayFastForward;
import kiel.editor.controller.ActionViewReplayPlay;
import kiel.editor.controller.ActionViewReplayRewind;
import kiel.editor.controller.ActionViewReplayStepBack;
import kiel.editor.controller.ActionViewReplayStepForward;
import kiel.editor.controller.ActionViewReplayStop;
import kiel.editor.controller.ActionViewReplayTraceLog;
import kiel.editor.controller.ActionViewShowCompositeStateEvents;
import kiel.editor.controller.ActionViewShowCompositeStateVariables;
import kiel.editor.controller.ActionViewShowStateActions;
import kiel.editor.controller.EditorAction;
import kiel.editor.controller.EditorActions;
import kiel.editor.controller.EditorModeAddDeepHistory;
import kiel.editor.controller.EditorModeAddDynamicChoice;
import kiel.editor.controller.EditorModeAddFinalSimpleState;
import kiel.editor.controller.EditorModeAddHistory;
import kiel.editor.controller.EditorModeAddHorizontalDelimiter;
import kiel.editor.controller.EditorModeAddInitialState;
import kiel.editor.controller.EditorModeAddOrState;
import kiel.editor.controller.EditorModeAddSimpleState;
import kiel.editor.controller.EditorModeAddSuspend;
import kiel.editor.controller.EditorModeAddVerticalDelimiter;
import kiel.editor.controller.EditorModeSelectOrDragOrResizeOrFlipExpand;
import kiel.editor.controller.EditorModes;
import kiel.editor.graph.CompositeStateCell;
import kiel.editor.resources.Preferences;
import kiel.editor.resources.ResourceBundle;
import kiel.editor.resources.ResourceLoader;
import kiel.graphicalInformations.View;
import kiel.util.LogFile;
import kiel.util.main.IKielFrame;
import kiel.util.main.KielComponent;

import org.jgraph.event.GraphSelectionEvent;
import org.jgraph.event.GraphSelectionListener;
import org.jgraph.graph.DefaultGraphModel;
```

```java
/**
 * The editor's architecture is based on the model-view-controller (MVC)
     design
 * pattern. The Editor class is the root class of the MVC's view component.
 * It contains all visible GUI-components: the drawing area (graph), the
     menubar
 * and the toolbar. The kiel.KielFrame class uses a single instance of the
 * Editor class and the Editor class itself has a reference to the KielFrame
     ,
 * since it will be the parent Frame for the Editor.
 * Mainly, the Editor just implements the KielComponent interface, to
     provide
 * the KielFrame its gui-components.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.143 $ last modified $Date: 2005/08/02 18:51:08 $
 */
public final class Editor implements KielComponent, Preferences.Listener,
    UndoClusterManagerListener {

    /**
     * The editor's graph drawing area. An Editor instance will be notified
     * with every graph's model change in order to update button states.
     */
    private MyJGraph graph = new MyJGraph(new DefaultGraphModel(), true);

    /**
     * Covers many (not all!) Editor's menu items. Take care, that this
     method
     * leaves some menu items disabled.
     * @param b enable the buttons?
     */
    private void setMenuItemsEnabled(final boolean b) {
        EditorActions.get(ActionFileSave.class).setEnabled(b);
        EditorActions.get(ActionFileSaveAs.class).setEnabled(b);
        EditorActions.get(ActionFileExport.class).setEnabled(b);
        EditorActions.get(ActionFilePrint.class).setEnabled(b);
//        EditorActions.get(ActionFileProperties.class).setEnabled(b);
        EditorActions.get(ActionEditEdit.class).setEnabled(b);
        EditorActions.get(ActionEditCut.class).setEnabled(b);
        EditorActions.get(ActionEditCopy.class).setEnabled(b);
        EditorActions.get(ActionEditPaste.class).setEnabled(b);
        EditorActions.get(ActionEditRemove.class).setEnabled(b);
        EditorActions.get(ActionEditSelectAll.class).setEnabled(b);
        EditorActions.get(ActionEditFind.class).setEnabled(b);
        EditorActions.get(ActionEditCollapsed.class).setEnabled(b);
        EditorModes.get(EditorModeAddSimpleState.class).setEnabled(b);
        EditorModes.get(EditorModeAddFinalSimpleState.class).setEnabled(b);
        EditorModes.get(EditorModeAddOrState.class).setEnabled(b);
        EditorModes.get(EditorModeAddInitialState.class).setEnabled(b);
        EditorModes.get(EditorModeAddSuspend.class).setEnabled(b);
        EditorModes.get(EditorModeAddHistory.class).setEnabled(b);
        EditorModes.get(EditorModeAddDynamicChoice.class).setEnabled(b);
        EditorModes.get(EditorModeAddHorizontalDelimiter.class).setEnabled(b
    );
        EditorModes.get(EditorModeAddVerticalDelimiter.class).setEnabled(b);
        EditorActions.get(ActionViewReplayFastForward.class).setEnabled(b);
        EditorActions.get(ActionViewReplayPlay.class).setEnabled(b);
        EditorActions.get(ActionViewReplayRewind.class).setEnabled(b);
        EditorActions.get(ActionViewReplayStepBack.class).setEnabled(b);
        EditorActions.get(ActionViewReplayStepForward.class).setEnabled(b);
        EditorActions.get(ActionViewReplayStop.class).setEnabled(b);
        graph.getUndoClusterManager().getSlider().setEnabled(b);
        EditorActions.get(ActionAddTransitionToggle.class).setEnabled(b);
        EditorActions.get(ActionFormatZoom100.class).setEnabled(b);
        EditorActions.get(ActionFormatZoomIn.class).setEnabled(b);
        EditorActions.get(ActionFormatZoomOut.class).setEnabled(b);
        EditorActions.get(ActionFormatAlignLeft.class).setEnabled(false);
        EditorActions.get(ActionFormatAlignRight.class).setEnabled(false);
        EditorActions.get(ActionFormatAlignTop.class).setEnabled(false);
        EditorActions.get(ActionFormatAlignBottom.class).setEnabled(false);
        EditorActions.get(ActionFormatAlignCenterHorizontal.class)
            .setEnabled(false);
        EditorActions.get(ActionFormatAlignCenterVertical.class)
            .setEnabled(false);
        EditorActions.get(ActionFormatAlignJustifyHorizontal.class)
            .setEnabled(false);
        EditorActions.get(ActionFormatAlignJustifyVertical.class)
            .setEnabled(false);
    }

    /**
     * if the parameters are not null, this method will set up the graph
     drawing
     * area (the graph) with a new model.
     * @see kiel.util.main.KielComponent#setCurrentStateChart(
     *      kiel.dataStructure.StateChart,
     *      kiel.graphicalInformations.View)
     */
    public void setCurrentStateChart(final StateChart statechart,
        final View view) {

        if (statechart != null && view != null) {
//            if (getGraph().getMyGraphModel() == null
//                || getGraph().getMyGraphModel().getCurrentStateChart()
//                    != statechart) {
                setModel(new MyGraphModel(statechart, view,
                    getGraph().getGraphLayoutCache()));
                getGraph().getMyGraphModel().processIsCollapsed(
                    (CompositeStateCell) getGraph().getMyGraphModel()
                        .getRootCell(), getGraph(), true);
                getGraph().getUndoClusterManager().discardCurrentCluster();
//            }
        } else {
            getGraph().setModel(
                new DefaultGraphModel());
        }
```

```java
}

/**
 * Returns the editor-icon displayed in the browser for the switching
 * button browser<-->editor.
 * @see kiel.util.main.KielComponent#getImageIcon()
 */
public ImageIcon getImageIcon() {
    return ResourceLoader.get("Edit16.gif");
}

/**
 * Delegates to the graph.
 * @see kiel.util.main.KielComponent#getCurrentStateChart()
 */
public StateChart getCurrentStateChart() {
    return getGraph().getCurrentStateChart();
}

/**
 * Delegates to the graph.
 * @see kiel.util.main.KielComponent#getCurrentView()
 */
public View getCurrentView() {
    return getGraph().getCurrentView();
}

/**
 * The status line component showing the cursor's coordinates.
 */
private JLabel labelCursorPosition = new JLabel();

/**
 * The status line component showing the current edit step count.
 */
private JLabel labelEditStepCount = new JLabel();

/**
 * @see kiel.editor.UndoClusterManagerListener#stackChanged(int, int)
 */
public void stackChanged(final int undoableEditSteps,
    final int redoableEditSteps) {
    labelEditStepCount.setText(
        undoableEditSteps + ":" + redoableEditSteps);
}

/**
 * Sets the text for the editor status line.
 * @param point current cursor position.
 */
public void setLabelCursorPosition(final Point2D point) {
    labelCursorPosition.setText(
        (int) point.getX() + ":" + (int) point.getY());
}

/**
 * Returns the label text area, the cursor coordinates area and the zoom
 * area of the status line as a single component.
 * @see kiel.util.main.KielComponent#getStatusLineComponent()
 */
public JComponent getStatusLineComponent() {

    JPanel statusLineComponent = Utils.createJPanel(
        new FlowLayout(FlowLayout.RIGHT, 0, 0));

    final JSlider slider = Utils.createJSlider();
        final int width = 200;
        final int height = 24;
    slider.setPreferredSize(new Dimension(width, height));
    slider.addChangeListener(new ChangeListener() {

        public void stateChanged(final ChangeEvent e) {
            final int middle = 50;
            getGraph().setScale(slider.getValue() / (double) middle);
        }
    });
    final int labelWidth = 50;
    final int labelHeight = 24;
    labelCursorPosition.setPreferredSize(
        new Dimension(labelWidth, labelHeight));

    statusLineComponent.add(labelEditStepCount);
    statusLineComponent.add(new JLabel("     "));
    statusLineComponent.add(labelCursorPosition);
    statusLineComponent.add(
        new JLabel("    " + ResourceBundle.getString("Zoom") + ":   "));
    statusLineComponent.add(
        new JLabel(ResourceLoader.get("ZoomOut16.gif")));
    statusLineComponent.add(slider);
    statusLineComponent.add(
        new JLabel(ResourceLoader.get("ZoomIn16.gif")));
    statusLineComponent.add(new JLabel("  "));

    return statusLineComponent;
}

/**
 * returns the editor's drawing area.
 * @return the editor's graph component.
 */
public MyJGraph getGraph() {
    return graph;
}

/**
 * The logging handle. Prints logs to standard out.
 */
private LogFile logFile = new LogFile("Editor", 2);
```

```java
/**
 *
 * @return .
 */
public LogFile getLogFile() {
    return logFile;
}

/**
 * Attachs itself as listener to the Preferences, initializes the
 toolbar's
 * comboboxes, sets all menu items to disabled, adds itself as listener
 * to the graph object to update some buttons and menu items depending
 on
 * whether a diagram element is selected or not.
 * @param aKielFrame .
 */
public Editor(final IKielFrame aKielFrame) {

    setKielFrame(aKielFrame);
    Preferences.addListener(this);
    EditorAction.initialize(this);
    setMenuItemsEnabled(false);
    graph.getUndoClusterManager().setListener(this);
    graph.setLayouterMenu(getKielFrame().createLayouterMenu(false,
        ActionLayoutSelectedState.class));

    menuLayout = getKielFrame().createLayouterMenu(true,
        ActionLayout.class);
    final int comboWidth = 70;
    final int comboHeight = 28;
    comboBoxTransitionLineTypes.setMaximumSize(
        new Dimension(comboWidth, comboHeight));
    comboBoxTransitionTypes.setMaximumSize(
        new Dimension(comboWidth, comboHeight));

    graph.addGraphSelectionListener(new GraphSelectionListener() {
        public void valueChanged(final GraphSelectionEvent e) {
            boolean enabled = graph.getSelectionCount() > 1;
            EditorActions.get(ActionFormatAlignLeft.class)
                .setEnabled(enabled);
            EditorActions.get(ActionFormatAlignRight.class)
                .setEnabled(enabled);
            EditorActions.get(ActionFormatAlignTop.class)
                .setEnabled(enabled);
            EditorActions.get(ActionFormatAlignBottom.class)
                .setEnabled(enabled);
            EditorActions.get(ActionFormatAlignCenterHorizontal.class)
                .setEnabled(enabled);
            EditorActions.get(ActionFormatAlignCenterVertical.class)
                .setEnabled(enabled);
            EditorActions.get(ActionFormatAlignJustifyHorizontal.class)
                .setEnabled(enabled);
            EditorActions.get(ActionFormatAlignJustifyVertical.class)
                .setEnabled(enabled);

//              updateMenuEdit();
        }
    });
    comboBoxTransitionTypes.setSelectedIndex(2);
}

/**
 * Update the "Edit"-menu.
 */
public void updateMenuEdit() {
    menuEdit.removeAll();
    menuEdit.add(Utils.createJMenuItem(ActionEditUndo.class));
    menuEdit.add(Utils.createJMenuItem(ActionEditRedo.class));
    menuEdit.addSeparator();
    menuEdit.add(Utils.createJMenuItem(ActionEditSelectAll.class));
    menuEdit.add(Utils.createJMenuItem(ActionEditFind.class));
    menuEdit.addSeparator();
    if (getGraph().getMyGraphModel() != null) {
        menuEdit.add(Utils.createJMenuItem(
            ActionEditStateChartInputEvents.class));
        menuEdit.add(Utils.createJMenuItem(
            ActionEditStateChartOutputEvents.class));
        menuEdit.addSeparator();
    }
    ArrayList allItems =
        ((MyJGraph) getGraph()).getMenuItemsForSelected();
    for (int i = 0; i < allItems.size(); i++) {
        if (allItems.get(i) instanceof JComponent) {
            menuEdit.add((JComponent) allItems.get(i));
        } else {
            menuEdit.add((EditorAction) allItems.get(i));
        }
    }
}

/**
 * Delegates to the graph and updates the toolbar.
 * Should be moved to MyJGraph.
 * @param model the new model to be set.
 */
public void setModel(final MyGraphModel model) {
    // @TODO move to MyJGraph and install listeners
    getGraph().setModel(model);
    updateToolBarAndMenuBar();
    updateMenuEdit();
    getKielFrame().resizeToolbarPanel();
    if (model instanceof MyGraphModel
        && getGraph().getOverviewGraph() != null) {

        getGraph().getOverviewGraph().setOpaque(true);
```

```
            for (int i = 0; i < menuLayout.getMenuComponents().length; i++)
    {
                menuLayout.getMenuComponents()[i].setEnabled(true);
            }
        }
    }

    /**
     * Just creates the replay toolbar with 6 buttons and the slider.
     * @return a new replay toolbar object.
     */
    private JToolBar createReplayToolBar() {
        JToolBar replayToolbar = Utils.createJToolBar();
        replayToolbar.add(Utils.createJButton(EditorActions.get(
            ActionViewReplayStop.class), "tooltipReplayStop"));
        replayToolbar.add(Utils.createJButton(EditorActions.get(
            ActionViewReplayPlay.class), "tooltipReplayPlay"));
        replayToolbar.add(Utils.createJButton(EditorActions.get(
            ActionViewReplayRewind.class), "tooltipReplayRewind"));
        replayToolbar.add(Utils.createJButton(EditorActions.get(
            ActionViewReplayStepBack.class), "tooltipReplayStepBack"));
        replayToolbar.add(Utils.createJButton(EditorActions.get(
            ActionViewReplayStepForward.class), "tooltipReplayStepForward"))
    ;
        replayToolbar.add(Utils.createJButton(EditorActions.get(
            ActionViewReplayFastForward.class), "tooltipReplayFastForward"))
    ;
        replayToolbar.add(getGraph().getUndoClusterManager().getSlider());
        return replayToolbar;
    }

    /**
     * Just creates the alignment toolbar with 8 buttons.
     * @return a new alignment toolbar object.
     */
    private JToolBar createAlignmentToolBar() {
        JToolBar alignmentToolbar = Utils.createJToolBar();
        alignmentToolbar.add(Utils.createJButton(EditorActions.get(
            ActionFormatAlignLeft.class), "tooltipAlignLeft"));
        alignmentToolbar.add(Utils.createJButton(EditorActions.get(
            ActionFormatAlignRight.class), "tooltipAlignRight"));
        alignmentToolbar.add(Utils.createJButton(EditorActions.get(
            ActionFormatAlignTop.class), "tooltipAlignTop"));
        alignmentToolbar.add(Utils.createJButton(EditorActions.get(
            ActionFormatAlignBottom.class), "tooltipAlignBottom"));
        alignmentToolbar.add(Utils.createJButton(EditorActions.get(
            ActionFormatAlignCenterHorizontal.class),
            "tooltipAlignCenterHorizontal"));
        alignmentToolbar.add(Utils.createJButton(EditorActions.get(
            ActionFormatAlignCenterVertical.class),
            "tooltipAlignCenterVertical"));
        alignmentToolbar.add(Utils.createJButton(EditorActions.get(
            ActionFormatAlignJustifyHorizontal.class),
            "tooltipJustifyHorizontal"));
        alignmentToolbar.add(Utils.createJButton(EditorActions.get(
            ActionFormatAlignJustifyVertical.class),
            "tooltipJustifyVertical"));
        return alignmentToolbar;
    }

    /**
     * Defines a JLayeredPane with some special behaviour: whenever the
     * component's bounds are changed, the containing child's bounds are
     changed
     * as well. This Layered pane will contain 3 layers: the splash screen,
     * the graph drawing area and the overview graph.
     */
    private JLayeredPane mainComponent = new JLayeredPane() {

        public void setBounds(final int x, final int y, final int width,
            final int height) {

            super.setBounds(x, y, width, height);
            final int layers = 3;
            for (int i = 0; i < layers; i++) {
                if (getComponentsInLayer(i).length != 0) {
                    getComponentsInLayer(i)[0].setBounds(0, 0, width, height
    );
                }
            }
        }
    };

    /**
     * Returns the layered pane decorated with a Scroll pane. this method
     * defines a mouse wheel listener for that scroll pane for handling
     * zoom and undo requests. Next, it provides a mechanism to update
     * the overview's scale, so that the overview always fits the window.
     *
     * @see kiel.util.main.KielComponent#getMainComponent()
     */
    public JComponent getMainComponent() {

        preferencesChanged();
        JScrollPane pane = new Utils.MyJScrollPane(getGraph()) {

            protected void processMouseWheelEvent(final MouseWheelEvent e) {
                if (e.isControlDown()) {
                    if (e.getWheelRotation() > 0) { // "down"
                        EditorActions.get(ActionFormatZoomIn.class)
                            .actionPerformed(null);
                    } else {
                        EditorActions.get(ActionFormatZoomOut.class)
                            .actionPerformed(null);
                    }
                } if (e.isShiftDown()) {
                    if (e.getWheelRotation() > 0) { // "down"
                        EditorActions.get(ActionEditUndo.class)
```

```java
                                .actionPerformed(null);
                    } else {
                        EditorActions.get(ActionEditRedo.class)
                            .actionPerformed(null);
                    }
                } else {
                    super.processMouseWheelEvent(e);
                }
            }
        };
        mainComponent.add(pane, new Integer(2));
        mainComponent.addComponentListener(new ComponentAdapter() {
            public void componentResized(final ComponentEvent e) {
                getGraph().updateOverviewGraphScale();
            }

        });

        return mainComponent;
    }

    /**
     * This notifies the editor with every change of the users preferences.
     * The editor will then check if still the splash screen and the
     overview
     * layer should be shown.
     * @see kiel.editor.resources.Preferences.Listener#preferencesChanged()
     */
    public void preferencesChanged() {

        for (int i = 0; i < 2; i++) {
            // TODO Why 2? --> constant or mainComponent.getLayers().length
            if (mainComponent.getComponentCountInLayer(i) > 0) {
                mainComponent.remove(mainComponent.getComponentsInLayer(i)
     [0]);
            }
        }

//        if (Preferences.getShowSplashScreen()) {
//            mainComponent.add(
//                new JLabel(ResourceLoader.get("splashscreen2.gif")),
//                new Integer(0));
//        } else {
        JPanel p = new JPanel();
        p.setBackground(Color.WHITE);
            mainComponent.add(p, new Integer(0));
//        }

        if (Preferences.getShowOverviewLayer()) {
            mainComponent.add(getGraph().getOverviewGraph(), new Integer(1))
    ;
        } else {
            p = new JPanel();
            p.setBackground(Color.WHITE);
            mainComponent.add(p, new Integer(1));
        }
        mainComponent.repaint();
    }

    /**
     * Just returns nothing (null).
     * @see kiel.util.main.KielComponent#getSubPanels()
     */
    public JTabbedPane getSubPanels() {
        return null;
    }

    /**
     * Returns nothing (null) since the editor does not have a tree panel.
     * @see kiel.util.main.KielComponent#getTreePanel()
     */
    public JComponent getTreePanel() {
        return null;
    }

    /**
     * Returns the main toolbar, the replay toolbar and the alignment
     toolbar.
     * @see kiel.util.main.KielComponent#getToolBars()
     */
    public ToolBar[] getToolBars() {
        return new ToolBar[]{
            new ToolBar(toolbar, ResourceBundle.getString("MainMenu"), true)
 ,
            new ToolBar(createReplayToolBar(),
                ResourceBundle.getString("ReplayMenu"),
                Preferences.getShowReplayToolBarOnLoad()),
            new ToolBar(createAlignmentToolBar(),
                ResourceBundle.getString("AlignmentMenu"),
                Preferences.getShowAlignmentToolBarOnLoad())};
    }

    /**
     * Returns the "New Esterel Studio" and "New Matlab" menu items.
     * @see kiel.util.main.KielComponent#getFileMenuItems1()
     */
    public JMenuItem[] getFileMenuItems1() {

        JMenuItem[] items = new JMenu[1];

        items[0] = Utils.createJMenu("ActionFileNew", "newMnemonic",
            "New16.gif");
        ((JMenu) items[0]).add(Utils.createJMenuItem(
            ActionFileNewEsterelStudio.class));
        ((JMenu) items[0]).add(Utils.createJMenuItem(
            ActionFileNewMatlab.class));
        return items;
    }
```

```java
/**
 * Returns the menu items: Close, Save, Save as, Export, Properties,
 Print
 * and Preferences.
 * @see kiel.util.main.KielComponent#getFileMenuItems2()
 */
public JMenuItem[] getFileMenuItems2() {
    return new JMenuItem[]{
        Utils.createJMenuItem(ActionFileSave.class),
        Utils.createJMenuItem(ActionFileSaveAs.class),
        Utils.createJMenuItem(ActionFileExport.class),
        Utils.createJMenuItem(ActionFilePrint.class),
        Utils.createJMenuItem(ActionFilePreferences.class)};
}

/**
 * Defines the menu item showing whether the currently selected state
 * is collapsed or not. for that purpose there exists a getter for that
 * variable to set it checked or not, used whenever the mouse is moved.
 */
private JMenuItem menuItemCollapsed = Utils.addAccelerator(
    new JCheckBoxMenuItem(EditorActions.get(ActionEditCollapsed.class)))
 ;

/**
 * Used by the EditorModeSelectOrDragOrResizeOrFlipExpand on every mouse
 * move event, indicating whether the currently selected state is
 collapsed
 * or not.
 * @return the menu item representing the collapsed state of the
 currently
 * selected state.
 */
public JMenuItem getMenuItemCollapsed() {
    return menuItemCollapsed;
}

/**
 * The content of this menu is dependent on what type of Statechat model
 * is used (Esterel studio or matlab).
 */
private JMenu menuEinfuegen =
    Utils.createJMenu("insert", "insertMnemonic");

/**
 * Creates a checkbox menu item and sets the selected state.
 * @param actionClass for which action a checkbox menu item should be
 * created?
 * @param selected the selected state of the new menu item.
 * @return a new menu item.
 */
private JMenuItem createJMenuItem(final Class actionClass,
    final boolean selected) {
    JCheckBoxMenuItem item =
        new JCheckBoxMenuItem(EditorActions.get(actionClass));
    item.setSelected(selected);
    return item;
}
/**
 * The contents of this menu change according to the currently selected
 * graph element.
 */
private JMenu menuEdit =
    Utils.createJMenu("Edit", "editMnemonic");

/**
 * The contents of this menu change according to the currently selected
 * graph element.
 */
private JMenu menuLayout;

/**
 * Returns all menus for the menubar between the "file-" and "help-"
 menu.
 * @see kiel.util.main.KielComponent#getRootMenus()
 */
public JMenu[] getRootMenus() {

    final int menuTotalCount = 6;
    int menuIndex = -1;
    JMenu[] items = new JMenu[menuTotalCount];

    items[++menuIndex] = menuEdit;

    updateMenuEdit();

    items[++menuIndex] =
        Utils.createJMenu("view", "viewMnemonic");
    items[menuIndex].add(createJMenuItem(ActionViewOverview.class,
        Preferences.getShowOverviewLayer()));
    items[menuIndex].add(createJMenuItem(ActionViewGrid.class,
        Preferences.getShowGrid()));
    items[menuIndex].add(createJMenuItem(
        ActionViewShowStateActions.class,
        Preferences.getShowStateActions()));
    items[menuIndex].add(createJMenuItem(
        ActionViewShowCompositeStateEvents.class,
        Preferences.getShowCompositeStateEvents()));
    items[menuIndex].add(createJMenuItem(
        ActionViewShowCompositeStateVariables.class,
        Preferences.getShowCompositeStateVariables()));

    items[++menuIndex] =
        Utils.createJMenu("ActionViewReplay", "Replay_Mnemonic");
    items[menuIndex].add(Utils.createJMenuItem(
        ActionViewReplayStop.class));
    items[menuIndex].add(Utils.createJMenuItem(
```

```java
            ActionViewReplayPlay.class));
        items[menuIndex].add(Utils.createJMenuItem(
            ActionViewReplayRewind.class));
        items[menuIndex].add(Utils.createJMenuItem(
            ActionViewReplayStepBack.class));
        items[menuIndex].add(Utils.createJMenuItem(
            ActionViewReplayStepForward.class));
        items[menuIndex].add(Utils.createJMenuItem(
            ActionViewReplayFastForward.class));
        items[menuIndex].add(Utils.createJMenuItem(
            ActionViewReplayTraceLog.class));

        items[++menuIndex] = menuLayout;

        items[++menuIndex] = menuEinfuegen;

        items[++menuIndex] = Utils.createJMenu("Format", "formatMnemonic");
        items[menuIndex].add(Utils.createJMenuItem(
            ActionFormatZoom100.class));
        items[menuIndex].add(Utils.createJMenuItem(
            ActionFormatZoomIn.class));
        items[menuIndex].add(Utils.createJMenuItem(
            ActionFormatZoomOut.class));
        items[menuIndex].addSeparator();
        items[menuIndex].add(Utils.createJMenuItem(
            ActionFormatAlignLeft.class));
        items[menuIndex].add(Utils.createJMenuItem(
            ActionFormatAlignRight.class));
        items[menuIndex].add(Utils.createJMenuItem(
            ActionFormatAlignTop.class));
        items[menuIndex].add(Utils.createJMenuItem(
            ActionFormatAlignBottom.class));
        items[menuIndex].add(Utils.createJMenuItem(
            ActionFormatAlignCenterHorizontal.class));
        items[menuIndex].add(Utils.createJMenuItem(
            ActionFormatAlignCenterVertical.class));
        items[menuIndex].add(Utils.createJMenuItem(
            ActionFormatAlignJustifyHorizontal.class));
        items[menuIndex].add(Utils.createJMenuItem(
            ActionFormatAlignJustifyVertical.class));

        return items;
    }

    /**
     * My toolbar.
     */
    private JToolBar toolbar = Utils.createJToolBar();

    /**
     * The toolbar's combo box for transition types.
     */
    private JComboBox comboBoxTransitionTypes =
        Utils.createJComboBox(new String[]{
            ResourceBundle.getString("Normal"),
            ResourceBundle.getString("Weak"),
            ResourceBundle.getString("Strong")});

    /**
     * The toolbar's combo box for transition line types.
     */
    private JComboBox comboBoxTransitionLineTypes =
        Utils.createJComboBox(new String[]{
            ResourceBundle.getString("Spline"),
            ResourceBundle.getString("Bezier"),
            ResourceBundle.getString("Orthogonal")});

    /**
     * sets the button representing the default editor mode, the selection
     * mode. (its the first button in the row).
     */
    public void setSelectionButtonSelected() {
        // this method is also called before the menu (and its selection
     button)
        // is initialized
        if (buttonSelection != null) {
            buttonSelection.setSelected(true);
        }
    }

    /**
     * the button representing the default editor mode, the selection mode.
     * (its the first button in the row).
     */
    private JToggleButton buttonSelection;

    /**
     * @return .
     */
    public int getCurrentTransitionLine() {
        return comboBoxTransitionLineTypes.getSelectedIndex();
    }

    /**
     * @return .
     */
    public int getCurrentTransitionType() {
        return comboBoxTransitionTypes.getSelectedIndex();
    }

    /**
     * @see kiel.util.main.KielComponent#getKielFrameTitle()
     */
    public String getKielFrameTitle() {
        return ResourceBundle.getString("KIEL_Statechart_Editor");
    }

    /**
```

```
     * Updates the toolbar and menubar depending on which statechart model
     * is currently used. this method next builds a button group for the
     editor
     * modes buttons (Selection, Add XY State).
     */
    private void updateToolBarAndMenuBar() {

        toolbar.removeAll();
        if (getGraph().getMyGraphModel() == null) {
            return;
        }
        buttonSelection = Utils.createJToggleButton(
            EditorModes.get(EditorModeSelectOrDragOrResizeOrFlipExpand.class
     ),
            "tooltipBackToSelectionMode");
        toolbar.add(buttonSelection);
        toolbar.addSeparator();
        toolbar.add(Utils.createJToggleButton(EditorModes.get(
            EditorModeAddInitialState.class),
            "tooltipAddInitialState"));
        if (isEsterelStudioType()) {
            toolbar.add(Utils.createJToggleButton(EditorModes.get(
                EditorModeAddDeepHistory.class),
                "tooltipAddDeepHistory"));
            toolbar.add(Utils.createJToggleButton(EditorModes.get(
                EditorModeAddSuspend.class),
                "tooltipAddSuspend"));
        } else {
            toolbar.add(Utils.createJToggleButton(EditorModes.get(
                EditorModeAddHistory.class),
                "tooltipAddHistory"));
        }
        toolbar.add(Utils.createJToggleButton(EditorModes.get(
            EditorModeAddDynamicChoice.class),
            "tooltipAddDynamicChoice"));
        toolbar.add(Utils.createJToggleButton(EditorModes.get(
            EditorModeAddSimpleState.class),
            "tooltipAddSimpleState"));
        toolbar.add(Utils.createJToggleButton(EditorModes.get(
            EditorModeAddFinalSimpleState.class),
            "tooltipAddFinalSimpleState"));
        toolbar.add(Utils.createJToggleButton(EditorModes.get(
            EditorModeAddOrState.class),
            "tooltipAddOrState"));
        toolbar.add(Utils.createJToggleButton(EditorModes.get(
            EditorModeAddHorizontalDelimiter.class),
            "tooltipAddHorizontalDelimiter"));
        toolbar.add(Utils.createJToggleButton(EditorModes.get(
            EditorModeAddVerticalDelimiter.class),
            "tooltipAddVerticalDelimiter"));

        // For the single-selection of creation-buttons...
        ButtonGroup bg = new ButtonGroup();
        for (int i = 0; i < toolbar.getComponents().length; i++) {
            if (toolbar.getComponents()[i] instanceof JToggleButton) {
                bg.add((JToggleButton) toolbar.getComponents()[i]);
            }
        }
        buttonSelection.setSelected(true);

        toolbar.addSeparator();
        toolbar.add(comboBoxTransitionTypes);
        toolbar.add(comboBoxTransitionLineTypes);
        toolbar.addSeparator();
        toolbar.add(Utils.createJButton(EditorActions.get(
            ActionAddTransitionToggle.class), "tooltipAddTransitionToggle"))
    ;
        toolbar.addSeparator();
        toolbar.add(Utils.createJButton(EditorActions.get(
            ActionEditUndo.class), "tooltipEditUndo"));
        toolbar.add(Utils.createJButton(EditorActions.get(
            ActionEditRedo.class), "tooltipEditRedo"));
        toolbar.addSeparator();
        toolbar.add(Utils.createJButton(EditorActions.get(
            ActionFormatZoom100.class), "tooltipFormatZoom100"));
        toolbar.add(Utils.createJButton(EditorActions.get(
            ActionFormatZoomIn.class), "tooltipFormatZommIn"));
        toolbar.add(Utils.createJButton(EditorActions.get(
            ActionFormatZoomOut.class), "tooltipFormatZoomOut"));
        toolbar.addSeparator();
        toolbar.add(Utils.createJButton(EditorActions.get(
            ActionEditEdit.class), "tooltipEditEdit"));
        toolbar.addSeparator();
        toolbar.add(Utils.createJButton(EditorActions.get(
            ActionEditRemove.class), "tooltipEditRemove"));

        setMenuItemsEnabled(true);

        menuEinfuegen.removeAll();
        menuEinfuegen.add(EditorModes.get(EditorModeAddSimpleState.class));
        menuEinfuegen.add(EditorModes.get(EditorModeAddFinalSimpleState.
    class));
        menuEinfuegen.add(EditorModes.get(EditorModeAddOrState.class));
        menuEinfuegen.addSeparator();
        menuEinfuegen.add(EditorModes.get(EditorModeAddInitialState.class));
        menuEinfuegen.add(EditorModes.get(EditorModeAddDynamicChoice.class))
    ;
        if (isEsterelStudioType()) {
            menuEinfuegen.add(EditorActions.get(
                EditorModeAddDeepHistory.class));
            menuEinfuegen.add(EditorModes.get(EditorModeAddSuspend.class));
        } else {
            menuEinfuegen.add(EditorActions.get(EditorModeAddHistory.class))
    ;
        }
        menuEinfuegen.addSeparator();
        menuEinfuegen.add(EditorModes.get(
            EditorModeAddHorizontalDelimiter.class));
```

```java
        menuEinfuegen.add(EditorModes.get(
            EditorModeAddVerticalDelimiter.class));
        menuEinfuegen.addSeparator();
        menuEinfuegen.add(EditorActions.get(ActionAddTransitionToggle.class)
     );
    }

    /**
     * Checks the model type.
     * @return true if the current model is a Estudio model.
     */
    private boolean isEsterelStudioType() {
        return ResourceBundle.getString("ActionFileNewEsterelStudio")
            .equals(getGraph().getMyGraphModel().getCurrentStateChart()
                .getModelSource());
    }

    /**
     * Reference to the main application window. Used as parent for dialog
     * windows.
     */
    private IKielFrame kielFrame;

    /**
     * Returns the main application window.
     * @return the main application window.
     */
    public IKielFrame getKielFrame() {
        return kielFrame;
    }

    /**
     * @see kiel.util.main.KielComponent#setKielFrame(kiel.util.main.
     IKielFrame)
     */
    public void setKielFrame(final IKielFrame aFrame) {
        kielFrame = aFrame;
    }
}
```

kiel.editor.EditStep

```java
package kiel.editor;

import kiel.editor.controller.EditorAction;

/**
 * Models an (undoable) edit step containing a timestamp and a reference to
 * the editor action. This is used for statistics (Trace log) and could be
 * used for a named undo history, even displayed in the undo menu item.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.15 $ last modified $Date: 2005/03/21 22:53:57 $
 */
public final class EditStep {

    /**
     * This is a ResourceBundle key representing that a edit step has been
     * started.
     */
    public static final String TYPE_START = "editStepTypeStart";

    /**
     * This is a ResourceBundle key representing that an edit step has been
     * ended successfully.
     */
    public static final String TYPE_SUCCESSFUL = "editStepTypeSuccessful";

    /**
     * This is a ResourceBundle key representing that a started edit step
     has
     * been cancelled.
     */
    public static final String TYPE_CANCELLED = "editStepTypeCancelled";

    /**
     * How many JGraph-framework-edit-steps are clustered with this edit
     step.
     */
    private int count;

    /**
     * Which EditorAction is represented by this EditStep.
     */
    private EditorAction editorAction;

    /**
     * When did this step occur.
     */
    private long currentTimeMillis;

    /**
     * Is this Edit step started, ended successfully or cancelled.
     */
    private String type;

    /**
     * Creates an EditStep, done in EditorAction, undoClusterManager and
     * EditorModeAddNode.
     * @param aCount @see count
     * @param anEditorAction @see editorAction
     * @param aType @see type
     */
    public EditStep(final int aCount, final EditorAction anEditorAction,
        final String aType) {

        count = aCount;
        editorAction = anEditorAction;
        currentTimeMillis = System.currentTimeMillis();
        type = aType;
    }

    /**
     * Returns the count of JGraph model changes clustered in this user edit
     * step.
     * @see #count
     * @return @see #count
     */
    public int getCount() {
        return count;
    }

    /**
     * Returns the EditorAction object which performed this user edit step.
     * @see #editorAction
     * @return @see #editorAction
     */
    public EditorAction getEditorAction() {
        return editorAction;
    }

    /**
     * When did this user edit step occur.
     * @see #currentTimeMillis
     * @return @see #currentTimeMillis
     */
    public long getCurrentTimeMillis() {
        return currentTimeMillis;
    }

    /**
     * Was the edit step successful or cancelled or whatever.
     * @see #type
```

```
     * @return @see #type
     */
    public String getType() {
        return type;
    }
}
```

kiel.editor.FileHistory

```java
package kiel.editor;

import java.awt.event.ActionEvent;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.Properties;
import java.util.Vector;
import javax.swing.AbstractAction;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.KeyStroke;

import kiel.editor.resources.Preferences;
import kiel.util.LogFile;
import kiel.util.main.IKielFrame;

/**
 * Organizes the file history, displayed in the file->Last files menu.
    Mainly
 * this class provides the JMenu containing the last files. This information
     is
 * stored in the user's home directory and updated whenever the add() method
     is
 * called.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.45 $ last modified $Date: 2005/07/25 11:44:37 $
 */
public final class FileHistory {

    /**
     * The file history is used in this application just as single object.
     */
    private static FileHistory instance;

    /**
     * just returns the single instance of this class.
     * @param aKielFrame Reference to the main application window.
     * @return just returns the single instance
     */
    public static FileHistory getInstance(final IKielFrame aKielFrame) {

        if (instance == null) {
            instance = new FileHistory(aKielFrame);
        }
        return instance;
    }

    /**
     * Used as handle for the file containing the file's list information.
     */
    private Properties properties = new Properties();

    /**
     * Reference to the application's main window, for using its file open
     * functionality.
     */
    private IKielFrame kielFrame;

    /**
     * The file in which the last opened file names are stored.
     */
    private static final String FILENAME =
        System.getProperty("user.home")
        + File.separator + ".kiel"
        + File.separator + "filelist";


    /**
     * The logging handle. Prints logs to standard out.
     */
    private LogFile logFile = new LogFile("Editor", 2);

    /**
     * The FileHistory needs a reference to the KielFrame in order to
     * use its open(File) method.
     * @param aKielFrame @see #kielFrame
     */
    private FileHistory(final IKielFrame aKielFrame) {

        kielFrame = aKielFrame;

        try {
            properties.load(new FileInputStream(FILENAME));
        } catch (IOException e) {
            logFile.log(0, "A new filelist will be created.");
            // Its ok. new environment...
        }
        for (int i = 0; i < Preferences.getLastFilesCount(); i++) {
            if (properties.get("" + i) != null) {
                addItem(i);
            }
        }
    }

    /**
     * Adds a menu item to the menu.
     * @param i the index of the item to be added.
     */
```

```java
private void addItem(final int i) {

    JMenuItem mi = new JMenuItem(new AbstractAction(new File(properties
            .getProperty("" + i)).getName(), null) {

        public void actionPerformed(final ActionEvent e) {
            kielFrame.open(new File(properties.getProperty("" + i)));
        }
    });
    final int maxAcceleratorCount = 9;
    if (i < maxAcceleratorCount) {
        mi.setAccelerator(KeyStroke.getKeyStroke((char) ('1' + i), 2));
    }
    lastOpenedFilesMenu.add(mi);
}

/**
 * Called by the editor whenever the user opens or stores a file.
 * This updates the list of recent used files.
 * @param file A new file to be added.
 */
public void add(final File file) {

    if (file == null) {
        return;
    }
    Vector list = new Vector();
    for (int i = 0; i < properties.size(); i++) {
        if (!list.contains(properties.getProperty("" + i))) {
            list.add(properties.getProperty("" + i));
        }
    }
    if (!list.contains(file.toString())) {
        list.add(0, file.toString());
    }
    while (list.size() > Preferences.getLastFilesCount()) {
        list.removeElementAt(Preferences.getLastFilesCount());
    }
    properties.clear();
    lastOpenedFilesMenu.removeAll();
    for (int i = 0; i < list.size(); i++) {
        properties.setProperty("" + i, (String) list.elementAt(i));
        addItem(i);
    }
    try {
        properties.store(new FileOutputStream(FILENAME), null);
    } catch (IOException ex) {
        kielFrame.handleException(
            ex.getMessage(), true, ex);
    }
}

/**
 * The main menu, provided for the KielFrame.
 */
private JMenu lastOpenedFilesMenu =
    Utils.createJMenu("lastFiles", "lastFilesMnemonic",
        "Open16.gif");

/**
 * Returns the main last files menu.
 * @return the main last files menu.
 */
public JMenu getMenu() {

    return lastOpenedFilesMenu;
}
}
```

kiel.editor.MorphingLayouter

```
package kiel.editor;

import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import kiel.dataStructure.ANDState;
import kiel.dataStructure.DelimiterLine;
import kiel.dataStructure.GraphicalObject;
import kiel.dataStructure.Node;
import kiel.dataStructure.Transition;
import kiel.editor.graph.DelimiterLineCell;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.graph.NodeCell;
import kiel.editor.graph.TransitionCell;
import kiel.editor.resources.Preferences;
import kiel.graphicalInformations.DelimiterLineLayoutInformation;
import kiel.graphicalInformations.NodeLayoutInformation;
import kiel.graphicalInformations.View;
import kiel.util.LogFile;

import org.jgraph.graph.AttributeMap;
import org.jgraph.graph.DefaultGraphCell;
import org.jgraph.graph.GraphConstants;
import org.jgraph.graph.Port;

/**
 * The MorphingLayouter takes a set of "current" coordinates and a
     corresponding
 * set of "target"-coordinates, searches for the largest delta beween
     current
 * and target, counted in pixels, and this largest delta will be the count n
      of
 * "delta-slices" the MorphingLayouter will produce. The Morphing Layouter
     will
 * then perform the change from current to target in exactly n steps. That
 * means, in every step at least one coordinate will change exactly at 1
     pixel,
 * and no other coordinate will change more than 1 pixel. All coordinates
     are
 * provided as floats, so most of them will change in a range between [0..1]
      in
 * every step.
 * In every step the calculated delta will be applied to the JGraph-objects
     and
 * a real change is performed on the JGraph-model by calling JGraphModel.
     edit().
 * The effect will be, that the complete delta (or change) will look like a
 * movie to the user with a speed which is adjustable though the Preferences
 * class. The Morphing Layouter even takes into account that this speed will
 * be different on every computer depending on its resources and processor
 * speed. So after 10 steps (that is also adjustable though the Preferences
 * class), the mechanism will check, if the current speed fits the desired
 * speed. That means, after each 10 (or more or less) steps it will reduce
     the
 * speed (by taking some sleep() calls) or enhance the speed by clustering
 * steps. Whenever a caller wants a change to be performed by the
 * MorphingLayouter, then all subsequent changes to the JGraph-model should
     be
 * done though the MorphingLayouter as well because the MorphingLayouter
 * ensures that all calls (all model changes) will be performed in the
     correct
 * order. That is why the constructor wants to know if a real morphing
     should
 * be done or just a one-step-change.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.79 $ last modified $Date: 2005/08/01 17:54:30 $
 */
public final class MorphingLayouter implements Runnable {

    /**
     * The logging handle. Prints logs to standard out.
     */
    private LogFile logFile = new LogFile("Editor", 2);

    /**
     * This should be implemented by a MorphingLayouter user, who wants to
     be
     * informed when his asynchronous call has finished.
     * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
     a>
     */
    public interface MyListener {

        /**
         * Will be called when the MorphingLayouter finished its work of one
         * "user"-call.
         */
        void morphingFinished();
    }

    /**
     * Reference to the current graph. That includes the current coordinates
     */
    private final MyJGraph sourceGraph;

    /**
```

```
     * contains a mapping graphCell -- Rectangle2D, containing the target
     * coordinates.
     */
    private HashMap targetMapping;

    /**
     * There is at most one listener to be informed about the end of the
     * MorphingLayouter call.
     */
    private MyListener listener;

    /**
     * @param aSourceGraph information about the current coordinates.
     * @param aListener usually the caller, who wants to be informed when
     the
     * asynchronous call will be terminated.
     * @param aDoMorphing should a real morphing be done or should the
     change
     * be performed in one single step.
     */
    public MorphingLayouter(final MyJGraph aSourceGraph,
        final MyListener aListener, final boolean aDoMorphing) {

        sourceGraph = aSourceGraph;
        listener = aListener;
        doMorphing = aDoMorphing;
    }

    /**
     * Is called in the public user-methods, ensuring all asynchronous
     * calls will be performed in the correct order.
     * @param morphingLayouter the instance to be processed.
     */
    private static synchronized void startThread(
        final MorphingLayouter morphingLayouter) {

        threadQueue.add(morphingLayouter);
    }

    /**
     * Ensures all MorphingLayouter calls will be processed in the correct
     * order.
     */
    private static MorphingLayouterQueue threadQueue =
        new MorphingLayouterQueue();

    /**
     * Provides a permanently running thread checking all 30 milliseconds if
     * there is a MorphingLayouter request to be performed. The problem is:
     * While processing a MorphingLayouter call, it is dangerous to do other
     * changes to the JGraph-model except for the changes that are processed
     * currently. Lets say coordinate x is changed in 10 steps from value 42
     * to value 52. Lets say the MorphingLayouter just performed the 4th
     step,
     * so the value is currently 46. If now different MorphingLayouter call
     * would change the value of x to the value 60, then this change would
     be
     * destroyed by the 5th step of the first MorphingLayouter call, which
     * will set the value to 47. ;-)
     *
     * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
     a>
     */
    private static class MorphingLayouterQueue extends Thread {

        /**
         * Immediately starts the MorphingLayouter after the class is loaded
     .
         */
        MorphingLayouterQueue() {
            start();
        }

        /**
         * This list contains MorphingLayouter objects.
         */
        private ArrayList queue = new ArrayList();

        /**
         * Take the first object out of the queue and remove it from the
     queue.
         * @return the first element of the queue.
         */
        public synchronized Runnable consumeNext() {
            return (MorphingLayouter) queue.remove(0);
        }

        /**
         * Append a new MorphingLayouter object to the queue.
         * @param runnable .
         */
        public synchronized void add(final Runnable runnable) {
            queue.add(runnable);
        }

        /**
         * Checks if at least one element is in the list.
         * @return true if the queue is not empty.
         */
        private synchronized boolean isFull() {
            return !queue.isEmpty();
        }

        /**
         * Checks every 30 milliseconds if there is a MorphingLayouter
     object
         * in the queue. if so, this object will be started synchronously on
         * this thread
```

```
     * @see java.lang.Runnable#run()
     */
    public void run() {
        while (true) {  // TODO Just run on request!
            try {
                if (isFull()) {
                    consumeNext().run();
                } else {
                    final int sleepingTime = 30;
                    sleep(sleepingTime);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

/**
 * Simply ensmalls the given graph cell.
 * @param cell contains the current coordinates.
 * @param targetBounds defines the target coordinates.
 */
public void collapseCompositeState(final DefaultGraphCell cell,
    final Rectangle2D targetBounds) {

    targetMapping = new HashMap();
    targetMapping.put(cell, targetBounds);
    startThread(this);
}

/**
 * Not just simply enlarges the given graph cell. In addition to that
 all
 * ancestors have to be moved to a probably new position, then maybe the
 * parent has to be resized. This resize has to be applied recursively!
 * @param cell contains the current coordinates.
 * @param targetBounds defines the target coordinates.
 */
public void expandCompositeState(final DefaultGraphCell cell,
    final Rectangle2D targetBounds) {

    targetMapping = new HashMap();
    targetMapping.put(cell, targetBounds);
    // recursion: root := aTargetMapping.keySet().iterator().next();
    // All ancestors have to be moved, the parent has to be resized,
 then
    // recursive: root := parent
    addCells(cell, targetBounds);
    startThread(this);
}

/**
 *
 * @param cell .
 * @param testCell .
 * @return .
 */
private static boolean isAncestorOrChildOfAncestor(
    final DefaultGraphCell cell, final DefaultGraphCell testCell) {
    if (cell == testCell.getParent()) {
        return false;
    } else if (cell.getParent() == testCell.getParent()) {
        return true;
    } else if (testCell.getParent() == null) {
        return false;
    } else {
        return isAncestorOrChildOfAncestor(
            cell, (DefaultGraphCell) testCell.getParent());
    }
}

/**
 * Is called by expandCompositeState() and recursively. Checks the
 * ancestors of the cell and the parent.
 * @param cell the cell whose ancestors and parent have to be checked.
 * @param targetBounds the cell's target bounds.
 */
private void addCells(final DefaultGraphCell cell,
    final Rectangle2D targetBounds) {

    final Rectangle2D sourceBounds =
        sourceGraph.getGraphLayoutCache().getMapping(
            cell, false).getBounds();
    if (sourceBounds.getX() != targetBounds.getX()
        || sourceBounds.getY() != targetBounds.getY()) {
        return;
    }
    final double deltaWidth = targetBounds.getWidth()
                            - sourceBounds.getWidth();
    final double deltaHeight = targetBounds.getHeight()
                             - sourceBounds.getHeight();
    if (deltaWidth < 0 && deltaHeight < 0) {
        // move nodes only on expand...
        return;
    }
    Rectangle2D parentBounds = null;
    if (cell.getParent() != null) {
        parentBounds = sourceGraph.getGraphLayoutCache()
            .getMapping(cell.getParent(), false).getBounds();
    }
    // The ancestors and its childs have to be moved...
    final Object[] all = sourceGraph.getDescendants(sourceGraph.getRoots
 ());
    double maxParentOverlapX = 0;
    double maxParentOverlapY = 0;
    for (int i = 0; i < all.length; i++) {
        if (all[i] instanceof NodeCell
```

```
        && isAncestorOrChildOfAncestor(cell, (DefaultGraphCell) all[
i])
        && all[i] != cell) {
        // its an ancestor...
        final Rectangle2D bounds =
            GraphConstants.getBounds(((DefaultGraphCell) all[i])
                .getAttributes()).getBounds2D();
        double overlapX = 0;
        double overlapY = 0;
        if (bounds.getX() > sourceBounds.getX()
                            + sourceBounds.getWidth()
            && bounds.getY()
                + bounds.getHeight() > sourceBounds.getY()) {
            // is right hand next to...
            if (bounds.getY() < targetBounds.getY()
                                + targetBounds.getHeight()) {
                // and will be captured (not right hand on the
bottom)
                overlapX = targetBounds.getX()
                           + targetBounds.getWidth()
                           + Preferences
                               .getMorphingLayouterBorderDistance()
                           - bounds.getX();
            }
            if (overlapX < 0) {
                overlapX = 0;
            }
        }
        if (bounds.getY() > sourceBounds.getY()
                            + sourceBounds.getHeight()
            && bounds.getX()
                + bounds.getWidth() > sourceBounds.getX()) {
            // is under
            if (bounds.getX() < targetBounds.getX()
                                + targetBounds.getWidth()) {
                // and will be captured (not right hand on the
bottom)
                overlapY = targetBounds.getY()
                           + targetBounds.getHeight()
                           + Preferences
                               .getMorphingLayouterBorderDistance()
                           - bounds.getY();
            }
            if (overlapY < 0) {
                overlapY = 0;
            }
        }
        if (overlapX > 0 || overlapY > 0) {
            if (!targetMapping.containsKey(all[i])) {
                targetMapping.put(
                    all[i],
                    new AttributeMap.SerializableRectangle2D(
                        bounds.getX() + overlapX,
                        bounds.getY() + overlapY,
                        bounds.getWidth(),
                        bounds.getHeight()));
            }
            if (parentBounds == null) {
                logFile.log(2, "parentBounds is null");
                continue;
            }
            // Check the parent bounds...
            double parentOverlapX =
                bounds.getX()
                + overlapX
                + bounds.getWidth()
                + Preferences.getMorphingLayouterBorderDistance()
                - parentBounds.getX()
                - parentBounds.getWidth();
            if (parentOverlapX < 0) {
                parentOverlapX = 0;
            }
            double parentOverlapY =
                bounds.getY()
                + overlapY
                + bounds.getHeight()
                + Preferences.getMorphingLayouterBorderDistance()
                - parentBounds.getY()
                - parentBounds.getHeight();
            if (parentOverlapY < 0) {
                parentOverlapY = 0;
            }
            if (parentOverlapX > maxParentOverlapX) {
                maxParentOverlapX = parentOverlapX;
            }
            if (parentOverlapY > maxParentOverlapY) {
                maxParentOverlapY = parentOverlapY;
            }
        }
    }
}
if (parentBounds != null) {
    double parentOverlapX = targetBounds.getX()
        + targetBounds.getWidth()
        + Preferences.getMorphingLayouterBorderDistance()
        - parentBounds.getX()
        - parentBounds.getWidth();
    double parentOverlapY = targetBounds.getY()
        + targetBounds.getHeight()
        + Preferences.getMorphingLayouterBorderDistance()
        - parentBounds.getY()
        - parentBounds.getHeight();

    if (parentOverlapX > maxParentOverlapX) {
        maxParentOverlapX = parentOverlapX;
    }
    if (parentOverlapY > maxParentOverlapY) {
        maxParentOverlapY = parentOverlapY;
```

```
            }
            if (maxParentOverlapX > 0 || maxParentOverlapY > 0) {
                // The parent has to be resized...
                if (!targetMapping.containsKey(cell.getParent())) {
                    targetMapping.put(cell.getParent(),
                        new AttributeMap.SerializableRectangle2D(
                            parentBounds.getX(),
                            parentBounds.getY(),
                            parentBounds.getWidth() + maxParentOverlapX,
                            parentBounds.getHeight() + maxParentOverlapY));
                }
                addCells((DefaultGraphCell) cell.getParent(),
                    (Rectangle2D) targetMapping.get(cell.getParent()));
            }
        }
    }

    /**
     * Is called only by applyNewView(). Iterates though the target view and
     * fills a hash map with graph cell -- Rectangle2D pairs.
     * @param targetView the view of which the target coordinates is taken
     * @return the filled hash map as equivalent to the target view.
     */
    private HashMap createTargetMapping(final View targetView) {

        HashMap mapping = new HashMap();
        DefaultGraphCell[] all = sourceGraph.getMyGraphModel().getAllCells()
   ;

        for (int i = 0; i < all.length; i++) {

            GraphicalObject graphicalObject = MyGraphConstants
            .getGraphicalObject(((DefaultGraphCell) all[i]).getAttributes()
   ;

            if (graphicalObject instanceof Node) {

                Point2D offset = MyGraphModelUtilities.calculateOffset(
                    (Node) graphicalObject,
                    targetView);
                NodeLayoutInformation layout = targetView
                    .getLayoutInformation((Node) graphicalObject);
                mapping.put(
                    all[i],
                    new AttributeMap.SerializableRectangle2D(
                        layout.getXPos() + offset.getX(),
                        layout.getYPos() + offset.getY(),
                        layout.getWidth(),
                        layout.getHeight()));

                if (graphicalObject instanceof ANDState) {

                    offset.setLocation(
                        offset.getX() + layout.getXPos(),
                        offset.getY() + layout.getYPos());

                    DelimiterLine[] delimiterLines = (DelimiterLine[])
                        ((ANDState) graphicalObject).getDelimiterLines()
                            .toArray(new DelimiterLine[0]);

                    for (int j = 0; j < delimiterLines.length; j++) {
                        DelimiterLineLayoutInformation delimiterLineLayout =
                            targetView.getLayoutInformation((DelimiterLine)
                                delimiterLines[j]);

                        Rectangle2D targetBounds =
                            new AttributeMap.SerializableRectangle2D(
                                delimiterLineLayout.getStart().getX()
                                    + offset.getX(),
                                delimiterLineLayout.getStart().getY()
                                    + offset.getY(),
                                delimiterLineLayout.getEnd().getX()
                                    - delimiterLineLayout.getStart().getX(),
                                delimiterLineLayout.getEnd().getY()
                                    - delimiterLineLayout.getStart().getY())
 ;
                        // TODO this check is done in MyGraphModel as well!
                        if (targetBounds.getWidth() == 0) {
                            targetBounds.setRect(targetBounds.getX(),
                                targetBounds.getY(), 1,
                                targetBounds.getHeight());
                        }
                        if (targetBounds.getHeight() == 0) {
                            targetBounds.setRect(targetBounds.getX(),
                                targetBounds.getY(), targetBounds.getWidth()
 ,
                                1);
                        }

                        mapping.put(sourceGraph.getMyGraphModel()
                            .getGraphCell(delimiterLines[j]), targetBounds);
                    }
                }

            } else if (graphicalObject instanceof Transition) {

                mapping.put(all[i],
                    MyGraphModelUtilities.
 convertTransitionPointsKielToJGraph(
                        (Transition) graphicalObject, targetView));
            }
        }
        return mapping;
    }

    /**
     * Calculates a target mapping out of the target view and then starts
     the
```

```
     * morphing.
     * @param targetView the target coordinates are taken from this object.
     */
    public void applyNewView(final View targetView) {
        targetMapping = createTargetMapping(targetView);
        startThread(this);
    }

    /**
     * Holds the maximum delta of points of the most changing coordinate.
     This
     * defines through how many steps the morphing should be processed.
     */
    private double steps;

    /**
     * Should a real morphing be performed? if no, then the change old-->new
     * will be done in one beat and so this MorphingLayouter won't do a real
     * morphing.
     */
    private boolean doMorphing;

    /**
     * This method processes the morphing: first calculates the deltas to be
     * performed with every step and second the iteration over the steps
     count
     * is performed. After every 10 steps (can be changed in Preferences)
     * the used time is checked to decide whether a sleep() or clustering of
     * steps is needed to ensure the correct speed of the algorithm.
     * @see java.lang.Runnable#run()
     */
    public synchronized void run() {

        final Object[] cells = targetMapping.keySet().toArray();
        final Map deltaMapping = calculateDeltas(cells);

        final long start = System.currentTimeMillis();
        int stepDelta = 1;
        long currentSleepingTime = 0;
        steps = Math.round(steps);

        if (Preferences.getMorphingLayouterStepInterval() > 0) {
            stepDelta = (int) (Preferences.getMorphingLayouterStepInterval()
                / cells.length);
            if (stepDelta == 0) {
                stepDelta = 1;
            }
        }
        if (!doMorphing) {
            stepDelta = (int) steps;
        }

        // Perform the morphing...
        // the "true" ensures that the last iteration will have
        // stepCount == steps
        for (int stepCount = 0; stepCount < steps;) {

            if (stepCount + stepDelta > steps) {
                stepDelta = (int) steps - stepCount;
                stepCount = (int) steps;
            } else {
                stepCount += stepDelta;
            }
            final boolean lastIteration = stepCount == steps;

            Map transportMap = new HashMap();

            for (int i = 0; i < cells.length; i++) {

                AttributeMap newAttributes = new AttributeMap();

                if (cells[i] instanceof Port
                    || deltaMapping.get(cells[i]) == null) {

                    continue;

                } else if (cells[i] instanceof NodeCell
                    || cells[i] instanceof DelimiterLineCell) {

                    Rectangle2D source = GraphConstants.getBounds(
                        ((DefaultGraphCell) cells[i]).getAttributes());
                    Rectangle2D delta = (Rectangle2D) deltaMapping.get(
                        cells[i]);

                    GraphConstants.setBounds(
                        newAttributes,
                        new AttributeMap.SerializableRectangle2D(
                            get(source.getX(), delta.getX(),
                                stepDelta, stepCount, lastIteration),
                            get(source.getY(), delta.getY(),
                                stepDelta, stepCount, lastIteration),
                            get(source.getWidth(), delta.getWidth(),
                                stepDelta, stepCount, lastIteration),
                            get(source.getHeight(), delta.getHeight(),
                                stepDelta, stepCount, lastIteration)));
//                  if (cells[i] instanceof DelimiterLineCell) {
//                      System.out.println(GraphConstants.getBounds(
//                          newAttributes));
//                  }

                } else if (cells[i] instanceof TransitionCell) {

                    Point2D[] source = (Point2D[]) GraphConstants.getPoints(
                        ((DefaultGraphCell) cells[i]).getAttributes())
                        .toArray(new Point2D[0]);
                    Point2D[] delta = (Point2D[]) deltaMapping.get(cells[i])
;
                    List target = new ArrayList();
```

```java
                for (int k = 0; k < source.length; k++) {
                    target.add(new AttributeMap.SerializablePoint2D(
                        get(source[k].getX(), delta[k].getX(),
                            stepDelta, stepCount, lastIteration),
                        get(source[k].getY(), delta[k].getY(),
                            stepDelta, stepCount, lastIteration)));
                }
                GraphConstants.setPoints(newAttributes, target);
            }
            transportMap.put(cells[i], newAttributes);
        }

        sourceGraph.getModel().edit(transportMap, null, null, null);

        if (stepCount == steps) {
            break;
        }

        if (Preferences.getMorphingLayouterStepInterval() == 0) {
            // Check the morphing speed...
            final double oneSecond = 1000;
            final int stepInterval =
                Preferences.getMorphingLayouterCheckInterval();
            final int sleepingTimeIncrease = 10;
            try {
                long tempSleepingTime =
                    (long) ((stepCount + 1)
                        / Preferences.getMorphingLayouterSpeed()
                        * oneSecond)
                        - (System.currentTimeMillis() - start);
//              if (stepCount % stepInterval == 0) {
                if (stepCount == stepInterval) {  // fit only once
                    if (tempSleepingTime
                        < -(oneSecond
                          / Preferences.getMorphingLayouterSpeed()
                           * (stepDelta + 1))) {
                          // we are too slow
                        stepDelta++;
                        if (currentSleepingTime > 0) {
                            currentSleepingTime = 0;
                        }
                    } else if (tempSleepingTime > oneSecond
                            / Preferences.getMorphingLayouterSpeed()
                            * (stepDelta - 1)
                                && stepDelta > 1) { // we are too fast
                        stepDelta--;
                        currentSleepingTime += sleepingTimeIncrease;
                    }
                }
                if (currentSleepingTime > 0 && stepDelta == 1) {
                    Thread.sleep(currentSleepingTime);
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
    listener.morphingFinished();
}

/**
 * Calculates for each node the bounds change to be performed and for
 each
 * transition the points change to be performed.
 * @param cells all nodes and transitions to be changed.
 * @return the deltaMapping.
 */
private Map calculateDeltas(final Object[] cells) {

    HashMap deltaMapping = new HashMap();

    for (int i = 0; i < cells.length; i++) {

        if (cells[i] instanceof Port) {

            // Skip!
            continue;

        } else if (cells[i] instanceof NodeCell) {

            deltaMapping.put(cells[i],
                calculateDelta(sourceGraph.getCellBounds(cells[i]),
                    (Rectangle2D) targetMapping.get(cells[i])));

        } else if (cells[i] instanceof DelimiterLineCell) {

            deltaMapping.put(cells[i],
                calculateDelta(sourceGraph.getCellBounds(cells[i]),
                    (Rectangle2D) targetMapping.get(cells[i])));

        } else if (cells[i] instanceof TransitionCell) {

            List source = GraphConstants.getPoints(
                ((DefaultGraphCell) cells[i]).getAttributes());
            List target = (List) targetMapping.get(cells[i]);

            // handle different points lengths...
            List sourceClone = (List) ((ArrayList) source).clone();
            sourceClone.remove(TransitionCell.DUMMY_POINT);
            sourceClone.remove(TransitionCell.DUMMY_POINT);

            // maybe some of the source points have to be deleted...
            while (target.size() < sourceClone.size()) {
                sourceClone.remove(0);
            }
            // maybe some new points have to be inserted in source...
            while (target.size() > sourceClone.size()) {
                sourceClone.add(target.get(0));
```

```java
                }
                AttributeMap newAttributes = new AttributeMap();
                GraphConstants.setPoints(newAttributes, sourceClone);
                HashMap transportMap = new HashMap();
                transportMap.put(cells[i], newAttributes);
                sourceGraph.getMyGraphModel().edit(
                    transportMap, null, null, null);

                // ...

                deltaMapping.put(cells[i],
                calculateDelta((Point2D[]) GraphConstants.getPoints(
                    ((DefaultGraphCell) cells[i]).getAttributes())
                        .toArray(new Point2D[0]),
                    (Point2D[]) target.toArray(new Point2D[0])));
            }
        }
        return deltaMapping;
    }

    /**
     * .
     * @param source the source points.
     * @param target the target points.
     * @return the delta.
     */
    private Point2D[] calculateDelta(
        final Point2D[] source, final Point2D[] target) {

        if (source == null || target == null) {
            return null;
        }
        Point2D[] delta = new Point2D[source.length];

        for (int i = 0; i < delta.length; i++) {
            if (Double.isInfinite(target[i].getX())
                || Double.isInfinite(target[i].getY())) {
                delta[i] = new AttributeMap.SerializablePoint2D(0, 0);
            } else {
                delta[i] = new AttributeMap.SerializablePoint2D(
                        target[i].getX() - source[i].getX(),
                        target[i].getY() - source[i].getY());
            }
            if (Math.abs(delta[i].getX()) > steps) {
                steps = Math.abs(delta[i].getX());
            }
            if (Math.abs(delta[i].getY()) > steps) {
                steps = Math.abs(delta[i].getY());
            }
        }
        return delta;
    }

    /**
     * .
     * @param source the source bounds.
     * @param target the target bounds.
     * @return the delta.
     */
    private Rectangle2D calculateDelta(
        final Rectangle2D source, final Rectangle2D target) {

        if (source == null || target == null) {
            return null;
        }

        Rectangle2D delta = new AttributeMap.SerializableRectangle2D(
                target.getX() - source.getX(),
                target.getY() - source.getY(),
                target.getWidth() - source.getWidth(),
                target.getHeight() - source.getHeight());
        if (Math.abs(delta.getX()) > steps) {
            steps = Math.abs(delta.getX());
        }
        if (Math.abs(delta.getY()) > steps) {
            steps = Math.abs(delta.getY());
        }
        if (Math.abs(delta.getWidth()) > steps) {
            steps = Math.abs(delta.getWidth());
        }
        if (Math.abs(delta.getHeight()) > steps) {
            steps = Math.abs(delta.getHeight());
        }
        return delta;
    }

    /**
     * calculates the delta for a particular coordinate depending on all
     these
     * parameters.
     * @param current the current value to be morphed.
     * @param delta the complete delta for all steps altogether
     * for this current value
     * @param stepDelta first it will be one. if no morphing is desired,
     then
     * this will be the value of steps.
     * @param currentStep How many steps have already been done?
     * @param doRound has the result to be rounded?
     * @return for how much the current value has to be morphed in the
     current
     * step?
     */
    private double get(final double current, final double delta,
        final int stepDelta, final double currentStep, final boolean doRound
     ) {

        double stepValue = delta * stepDelta / steps;
```

```
    double result = current - (currentStep * stepValue) + ((currentStep
+ 1)
        * stepValue);
    if (doRound) {
        return Math.round(result);
    } else {
        return result;
    }
  }
}
```

kiel.editor.MyBasicGraphUI

```
package kiel.editor;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;

import kiel.editor.controller.ActionEditEdit;
import kiel.editor.controller.EditorActions;
import kiel.editor.controller.EditorModeAddNode;
import kiel.editor.controller.EditorModeAddTransition;
import kiel.editor.controller.EditorModeSelectOrDragOrResizeOrFlipExpand;
import kiel.editor.controller.EditorModes;
import kiel.editor.graph.Previewable;

import org.jgraph.JGraph;
import org.jgraph.graph.AttributeMap;
import org.jgraph.graph.CellHandle;
import org.jgraph.graph.CellView;
import org.jgraph.graph.GraphContext;
import org.jgraph.plaf.basic.BasicGraphUI;

/**
 * The BasicGraphUI class of the JGraph framework is a hook to change the
 * behaviour of mouse events and drawing child elements for example.
 * MyBasicGraphUI defines a speical handling for drawing child elements,
    since
 * JGraph does not know statechart hierarchy. then there is also set the
    kind
 * of drawing: turning on an antialiasing for smooth effects and higher
 * rendering quality. this is done directly on the Graphics object.
 * The mouse handlers defined in this class control the popup menu, the
 * changing of the Editor Modes (core feature!), and the delegating of mouse
 * events to the EditorMode objects. The MouseHandler also turns of the
 * JGraphs legacy kind of element selection. The RootHandle defined in this
 * class handles some special behaviour concerning the preview feature.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.51 $ last modified $Date: 2005/07/24 23:02:51 $
 */
class MyBasicGraphUI extends BasicGraphUI {

    /**
     * Just defines a mechanism to store the current preview bounds (while
     * dragging) in the belonging cell view. This information is needed for
     * checking whether the current preview position is valid or invalid
     * (red cross on the cursor).
     * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
     a>
     */
    private class MyRootHandle extends RootHandle {

        /**
         * Constructs an instance by just calling the super constructor.
         * @param context the graph context.
         */
        MyRootHandle(final GraphContext context) {
            super(context);
        }

        /**
         * Read the class description.
         * @see org.jgraph.graph.CellHandle#mouseDragged(
         * java.awt.event.MouseEvent)
         */
        public void mouseDragged(final MouseEvent e) {
            super.mouseDragged(e);

            for (int i = 0; i < views.length; i++) {
                CellView movingView = getGraph().getGraphLayoutCache()
                    .getMapping(views[i].getCell(), false);
                if (movingView instanceof Previewable) {
                    if (!((Previewable) movingView).isResizing()) {
                        ((Previewable) movingView).setPreviewBounds(
                            views[i].getBounds());
                    }
                }
            }
        }
    }

    /**
     * Returns the associated JGraph object.
     * @return the associated JGraph object.
     */
    protected JGraph getGraph() {
        return graph;
    }
    /**
     * The hook for defining an own MouseHandler.
     * @see org.jgraph.plaf.basic.BasicGraphUI#createMouseListener()
     */
    protected MouseListener createMouseListener() {
        return new MyMouseHandler();
    }

    /**
```

```java
     * This is the central class for the EditorMode-classes. The
     MyMouseHandler
     * receives mouse events from the graph surface and delegates them to
     the
     * current EditorMode object. In addition at mousePressed a new
     * EditorMode is set, at mouseReleased the popup menu is shown.
     * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
     a>
     */
    private class MyMouseHandler extends MouseHandler {

        /**
         * Delegates the event to the current EditorMode.
         * @see java.awt.event.MouseMotionListener#mouseMoved(
         * java.awt.event.MouseEvent)
         */
        public void mouseMoved(final MouseEvent e) {
            super.mouseMoved(e);
            EditorModes.getCurrent().mouseMoved(
                getGraph().fromScreen(e.getPoint()));
        }

        /**
         * Delegates the event to the current EditorMode.
         * @see java.awt.event.MouseMotionListener#mouseDragged(
         * java.awt.event.MouseEvent)
         */
        public void mouseDragged(final MouseEvent e) {
            super.mouseDragged(e);
            EditorModes.getCurrent().mouseDragged(
                getGraph().fromScreen(e.getPoint()));
        }

        /**
         * Delegates the event to the current EditorMode.
         * In addition a new EditorMode is set.
         * @see java.awt.event.MouseListener#mousePressed(
         * java.awt.event.MouseEvent)
         */
        public void mousePressed(final MouseEvent e) {

            if (e.isPopupTrigger() && !e.isShiftDown()) {
                handlePopupTrigger(e);
            } else {
                super.mousePressed(e);

                // correct the selection done in super.mousePressed
                CellView selected = graph.getGraphLayoutCache().getMapping(
                        graph.getSelectionCell(), false);
                if (!e.isControlDown() && selected != null) {
                    cell = selected;
                }
                // ...
                Point2D point = getGraph().fromScreen(e.getPoint());

                if (!(EditorModes.getCurrent() instanceof EditorModeAddNode)
) {

                    if (getGraph().getSelectionCell() != null) {

                        getGraph().setJumpToDefaultPort(false);
                        // JGraph framework
                        // always switches to "true" back! That produces a
wrong
                        // getPortViewAt() result!
                        if (getGraph().getPortViewAt(point.getX(), point.
getY())
                            != null && getGraph().isPortsVisible()) {

                            EditorModes.get(
                                EditorModeAddTransition.class).start();
                        }
                    } else {
                        EditorModes.get(
                            EditorModeSelectOrDragOrResizeOrFlipExpand.class
)
                            .start();
                    }
                }
                EditorModes.getCurrent().mousePressed(point, e.isControlDown
());
            }
        }

        /**
         * Opens the context menu.
         * @param e the corresponding event.
         */
        private void handlePopupTrigger(final MouseEvent e) {

            ((MyJGraph) getGraph()).setCurrentPopupPoint(e.getPoint());
            // correct right mouse clicks (these are only popup triggers, no
            // selection change!)
            CellView view =
                MyGraphModelUtilities.getSmallestViewOfType(
                    CellView.class,
                    new AttributeMap.SerializableRectangle2D(
                        ((MyJGraph) getGraph()).getCurrentPopupPoint().getX
(),
                        ((MyJGraph) getGraph()).getCurrentPopupPoint().getY
(),
                        1, 1), getGraph().getGraphLayoutCache(), true,
                        null);
            if (view == null) {
                return;    // Happens when no statechart is displayed
            }
            Object cell = view.getCell();
```

```java
        if (cell != null && getGraph().getSelectionCell()
            != cell) {

            getGraph().setSelectionCell(cell);
            ((MyJGraph) getGraph()).setCurrentPopupPoint(null);
            return;
        }
        ((MyJGraph) getGraph()).showPopup(e.getX(), e.getY());
    }

    /**
     * Delegates the event to the current EditorMode.
     * In addion the popup menu is shown if needed.
     * @see java.awt.event.MouseListener#mouseReleased(
     * java.awt.event.MouseEvent)
     */
    public void mouseReleased(final MouseEvent e) {

        if (e.isPopupTrigger() && !e.isShiftDown()) {
            handlePopupTrigger(e);
        } else {
            super.mouseReleased(e); // should not be called when
            // isPopupTrigger. otherwise the selection would be changed!
            EditorModes.getCurrent().mouseReleased(
                getGraph().fromScreen(e.getPoint()));
        }
    }

    /**
     * Just do nothing if the cell is selected.
     * @see org.jgraph.plaf.basic.BasicGraphUI.MouseHandler
     * #postProcessSelection(java.awt.event.MouseEvent,
     * java.lang.Object, boolean)
     */
    protected void postProcessSelection(final MouseEvent e,
        final Object cell, final boolean wasSelected) {
    }
}

/**
 * The hook for defining an own RootHandle.
 * @see org.jgraph.plaf.basic.BasicGraphUI
 * #createHandle(org.jgraph.graph.GraphContext)
 */
public CellHandle createHandle(final GraphContext context) {

    if (context != null && !context.isEmpty() && getGraph().isEnabled())
  {
        return new MyRootHandle(context);
    } else {
        return null;
    }
}

/**
 * Defines a special handling for drawing child elements, since
 * JGraph does not know statechart hierarchy. then there is also set the
 * kind of drawing: turning on an antialiasing for smooth effects and
 * higher rendering quality. this is done directly on the Graphics
 object.
 * @see org.jgraph.plaf.GraphUI#paintCell(java.awt.Graphics,
 *      org.jgraph.graph.CellView, java.awt.geom.Rectangle2D,
 *      boolean)
 */
public void paintCell(final Graphics g, final CellView view,
    final Rectangle2D bounds, final boolean preview) {

    RenderingHints renderHints = new RenderingHints(
        RenderingHints.KEY_ANTIALIASING,
        RenderingHints.VALUE_ANTIALIAS_ON);
    renderHints.put(
        RenderingHints.KEY_RENDERING,
        RenderingHints.VALUE_RENDER_QUALITY);
    ((Graphics2D) g).setRenderingHints(renderHints);
    super.paintCell(g, view, bounds, preview);
    // Then Paint Children
    CellView[] children = view.getChildViews();
    for (int i = 0; i < children.length; i++) {
        paintCell(g, children[i], children[i].getBounds(), preview);
    }
}

/**
 * @see org.jgraph.plaf.basic.BasicGraphUI#startEditing(
 * java.lang.Object, java.awt.event.MouseEvent)
 */
protected boolean startEditing(final Object cell, final MouseEvent event
 ) {
    EditorActions.get(ActionEditEdit.class).actionPerformed(null);
    return true;
}
}
```

kiel.editor.MyDefaultCellViewFactory

```java
package kiel.editor;

import kiel.editor.graph.ANDStateCell;
import kiel.editor.graph.ANDStateView;
import kiel.editor.graph.ChoiceCell;
import kiel.editor.graph.ChoiceView;
import kiel.editor.graph.ConditionalTransitionCell;
import kiel.editor.graph.ConditionalTransitionView;
import kiel.editor.graph.DeepHistoryCell;
import kiel.editor.graph.DeepHistoryView;
import kiel.editor.graph.DelimiterLineCell;
import kiel.editor.graph.DelimiterLineView;
import kiel.editor.graph.DynamicChoiceCell;
import kiel.editor.graph.DynamicChoiceView;
import kiel.editor.graph.FinalANDStateCell;
import kiel.editor.graph.FinalANDStateView;
import kiel.editor.graph.FinalORStateCell;
import kiel.editor.graph.FinalORStateView;
import kiel.editor.graph.FinalSimpleStateCell;
import kiel.editor.graph.FinalSimpleStateView;
import kiel.editor.graph.FinalStateCell;
import kiel.editor.graph.FinalStateView;
import kiel.editor.graph.ForkConnectorCell;
import kiel.editor.graph.ForkConnectorView;
import kiel.editor.graph.HistoryCell;
import kiel.editor.graph.HistoryView;
import kiel.editor.graph.InitialArcCell;
import kiel.editor.graph.InitialArcView;
import kiel.editor.graph.InitialStateCell;
import kiel.editor.graph.InitialStateView;
import kiel.editor.graph.JoinCell;
import kiel.editor.graph.JoinView;
import kiel.editor.graph.JunctionCell;
import kiel.editor.graph.JunctionView;
import kiel.editor.graph.NormalTerminationCell;
import kiel.editor.graph.NormalTerminationView;
import kiel.editor.graph.ORStateCell;
import kiel.editor.graph.ORStateView;
import kiel.editor.graph.RegionCell;
import kiel.editor.graph.RegionView;
import kiel.editor.graph.SimpleStateCell;
import kiel.editor.graph.SimpleStateView;
import kiel.editor.graph.StateCell;
import kiel.editor.graph.StateView;
import kiel.editor.graph.StrongAbortionCell;
import kiel.editor.graph.StrongAbortionView;
import kiel.editor.graph.SuspendCell;
import kiel.editor.graph.SuspendView;
import kiel.editor.graph.SuspensionCell;
import kiel.editor.graph.SuspensionView;
import kiel.editor.graph.SynchStateCell;
import kiel.editor.graph.SynchStateView;
import kiel.editor.graph.TransitionCell;
import kiel.editor.graph.TransitionView;
import kiel.editor.graph.WeakAbortionCell;
import kiel.editor.graph.WeakAbortionView;

import org.jgraph.graph.DefaultCellViewFactory;
import org.jgraph.graph.EdgeView;
import org.jgraph.graph.VertexView;

/**
 * The factory for Cell views, used by MyJGraph.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.7 $ last modified $Date: 2005/03/21 22:53:57 $
 */
class MyDefaultCellViewFactory extends DefaultCellViewFactory {

    /**
     * @see org.jgraph.graph.DefaultCellViewFactory#createEdgeView(
     * java.lang.Object)
     */
    protected EdgeView createEdgeView(final Object cell) {

        if (cell.getClass() == ConditionalTransitionCell.class) {
            return new ConditionalTransitionView(cell);
        } else if (cell.getClass() == InitialArcCell.class) {
            return new InitialArcView(cell);
        } else if (cell.getClass() == NormalTerminationCell.class) {
            return new NormalTerminationView(cell);
        } else if (cell.getClass() == SuspensionCell.class) {
            return new SuspensionView(cell);
        } else if (cell.getClass() == TransitionCell.class) {
            return new TransitionView(cell);
        } else if (cell.getClass() == StrongAbortionCell.class) {
            return new StrongAbortionView(cell);
        } else if (cell.getClass() == WeakAbortionCell.class) {
            return new WeakAbortionView(cell);
        } else {
            return null;
        }
    }

    /**
     * @see org.jgraph.graph.DefaultCellViewFactory#createVertexView(
     * java.lang.Object)
     */
    protected VertexView createVertexView(final Object cell) {

        if (cell.getClass() == ANDStateCell.class) {
```

```
      return new ANDStateView(cell);
    } else if (cell.getClass() == ChoiceCell.class) {
      return new ChoiceView(cell);
    } else if (cell.getClass() == DeepHistoryCell.class) {
      return new DeepHistoryView(cell);
    } else if (cell.getClass() == DynamicChoiceCell.class) {
      return new DynamicChoiceView(cell);
    } else if (cell.getClass() == FinalANDStateCell.class) {
      return new FinalANDStateView(cell);
    } else if (cell.getClass() == FinalORStateCell.class) {
      return new FinalORStateView(cell);
    } else if (cell.getClass() == FinalSimpleStateCell.class) {
      return new FinalSimpleStateView(cell);
    } else if (cell.getClass() == FinalStateCell.class) {
      return new FinalStateView(cell);
    } else if (cell.getClass() == ForkConnectorCell.class) {
      return new ForkConnectorView(cell);
    } else if (cell.getClass() == HistoryCell.class) {
      return new HistoryView(cell);
    } else if (cell.getClass() == InitialStateCell.class) {
      return new InitialStateView(cell);
    } else if (cell.getClass() == JoinCell.class) {
      return new JoinView(cell);
    } else if (cell.getClass() == JunctionCell.class) {
      return new JunctionView(cell);
    } else if (cell.getClass() == ORStateCell.class) {
      return new ORStateView(cell);
    } else if (cell.getClass() == RegionCell.class) {
      return new RegionView(cell);
    } else if (cell.getClass() == SimpleStateCell.class) {
      return new SimpleStateView(cell);
    } else if (cell.getClass() == StateCell.class) {
      return new StateView(cell);
    } else if (cell.getClass() == SuspendCell.class) {
      return new SuspendView(cell);
    } else if (cell.getClass() == SynchStateCell.class) {
      return new SynchStateView(cell);
    } else if (cell.getClass() == DelimiterLineCell.class) {
      return new DelimiterLineView(cell);
    } else {
      return null;
    }
  }
}
```

kiel.editor.MyGraphModel

```
package kiel.editor;

import java.awt.Dimension;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.lang.reflect.InvocationTargetException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collection;
import java.util.Collections;
import java.util.Comparator;
import java.util.HashMap;
import java.util.Hashtable;
import java.util.List;
import java.util.Map;

import javax.swing.undo.UndoableEdit;

import kiel.dataStructure.ANDState;
import kiel.dataStructure.Choice;
import kiel.dataStructure.CompositeState;
import kiel.dataStructure.CompoundLabel;
import kiel.dataStructure.ConditionalTransition;
import kiel.dataStructure.DeepHistory;
import kiel.dataStructure.DelimiterLine;
import kiel.dataStructure.DynamicChoice;
import kiel.dataStructure.FinalANDState;
import kiel.dataStructure.FinalORState;
import kiel.dataStructure.FinalSimpleState;
import kiel.dataStructure.FinalState;
import kiel.dataStructure.ForkConnector;
import kiel.dataStructure.GraphicalObject;
import kiel.dataStructure.History;
import kiel.dataStructure.InitialArc;
import kiel.dataStructure.InitialState;
import kiel.dataStructure.Join;
import kiel.dataStructure.Junction;
import kiel.dataStructure.Node;
import kiel.dataStructure.NormalTermination;
import kiel.dataStructure.ORState;
import kiel.dataStructure.PseudoState;
import kiel.dataStructure.Region;
import kiel.dataStructure.SimpleState;
import kiel.dataStructure.State;
import kiel.dataStructure.StateChart;
import kiel.dataStructure.StrongAbortion;
import kiel.dataStructure.Suspend;
import kiel.dataStructure.Suspension;
import kiel.dataStructure.SynchState;
import kiel.dataStructure.Transition;
import kiel.dataStructure.WeakAbortion;
import kiel.editor.graph.ANDStateCell;
import kiel.editor.graph.ChoiceCell;
import kiel.editor.graph.CompositeStateCell;
import kiel.editor.graph.CompositeStateView;
import kiel.editor.graph.ConditionalTransitionCell;
import kiel.editor.graph.DeepHistoryCell;
import kiel.editor.graph.DelimiterLineCell;
import kiel.editor.graph.DynamicChoiceCell;
import kiel.editor.graph.FinalANDStateCell;
import kiel.editor.graph.FinalORStateCell;
import kiel.editor.graph.FinalSimpleStateCell;
import kiel.editor.graph.FinalStateCell;
import kiel.editor.graph.ForkConnectorCell;
import kiel.editor.graph.HistoryCell;
import kiel.editor.graph.InitialArcCell;
import kiel.editor.graph.InitialStateCell;
import kiel.editor.graph.JoinCell;
import kiel.editor.graph.JunctionCell;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.graph.NodeCell;
import kiel.editor.graph.NormalTerminationCell;
import kiel.editor.graph.ORStateCell;
import kiel.editor.graph.PseudoStateCell;
import kiel.editor.graph.RegionCell;
import kiel.editor.graph.SimpleStateCell;
import kiel.editor.graph.StateCell;
import kiel.editor.graph.StrongAbortionCell;
import kiel.editor.graph.SuspendCell;
import kiel.editor.graph.SuspensionCell;
import kiel.editor.graph.SynchStateCell;
import kiel.editor.graph.TransitionCell;
import kiel.editor.graph.WeakAbortionCell;
import kiel.editor.resources.Preferences;
import kiel.editor.resources.ResourceBundle;
import kiel.graphicalInformations.CompositeStateLayoutInformation;
import kiel.graphicalInformations.DelimiterLineLayoutInformation;
import kiel.graphicalInformations.EdgeLayoutInformation;
import kiel.graphicalInformations.LabelLayoutInformation;
import kiel.graphicalInformations.LinetoPath;
import kiel.graphicalInformations.NodeLayoutInformation;
import kiel.graphicalInformations.Point;
import kiel.graphicalInformations.Properties;
import kiel.graphicalInformations.View;
import kiel.util.CompoundLabelException;
import kiel.util.CompoundLabelParser;
import kiel.util.LogFile;

import org.jgraph.graph.AttributeMap;
import org.jgraph.graph.CellView;
import org.jgraph.graph.ConnectionSet;
import org.jgraph.graph.DefaultGraphCell;
```

```java
import org.jgraph.graph.DefaultGraphModel;
import org.jgraph.graph.DefaultPort;
import org.jgraph.graph.Edge;
import org.jgraph.graph.GraphCell;
import org.jgraph.graph.GraphConstants;
import org.jgraph.graph.GraphLayoutCache;
import org.jgraph.graph.ParentMap;
import org.jgraph.graph.Port;

/**
 * MyGraphModel represents the model in the MVC pattern. Mainly it contains
 * functions for transforming between the KIEL datatstructure and the
 * JGraph datastructure. The MyGraphModel contains the complete KIEL
 * data model as well.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.104 $ last modified $Date: 2005/08/02 19:27:55 $
 */
public final class MyGraphModel extends DefaultGraphModel {

    /**
     * This mapping is used for building up the transitions in the new
     * JGraph-Tree. This is needed since jgraph transitions are connected to
     * a parent cell, but kiel transitions are just connected by source and
     * target.
     */
    private Hashtable nodesToCells = new Hashtable();

    /**
     * looks up the JGraph graph cell object for the KIEL graphical object.
     * @param graphicalObject the look up key.
     * @return the corresponding graph cell.
     */
    public DefaultGraphCell getGraphCell(
        final GraphicalObject graphicalObject) {
        return (DefaultGraphCell) nodesToCells.get(graphicalObject);
    }

    /**
     * The root for topological information of the KIEL datastructure.
     */
    private StateChart kielStatechart;

    /**
     * The root for graphical informations of the KIEL datastructure.
     */
    private View kielView;

    /**
     * The root of the JGraph datastructure..
     */
    private DefaultGraphCell statechartRootCell;

    /**
     * contains the real transition bounds.
     */
    private GraphLayoutCache graphLayoutCache;

    /**
     * Returns the model's root cell.
     * @return the model's root cell.
     */
    public DefaultGraphCell getRootCell() {
        return statechartRootCell;
    }

    /**
     * The logging handle. Prints logs to standard out.
     */
    private static LogFile logFile = new LogFile("Editor", 2);

    /**
     * Returns all cells of this model.
     * @return all cells of this model.
     */
    public DefaultGraphCell[] getAllCells() {
        return getAllCells(statechartRootCell);
    }

    /**
     * Checks if a transition's source is ok.
     * @param edge is not interpreted.
     * @param port checks for the parent of this port, if it accepts
     outgoing
     * transitions.
     * @return false if the port belongs to a history cell, a deepHistory
     cell
     * or the root cell.
     * @see org.jgraph.graph.GraphModel#acceptsSource(java.lang.Object,
     *      java.lang.Object)
     */
    public boolean acceptsSource(final Object edge, final Object port) {
        return port != null
                && ((DefaultPort) port).getParent().getClass()
                        != FinalStateCell.class
                && ((DefaultPort) port).getParent().getClass()
                        != FinalSimpleStateCell.class
                && ((DefaultPort) port).getParent().getClass()
                        != FinalANDStateCell.class
                && ((DefaultPort) port).getParent().getClass()
                        != FinalORStateCell.class
                && ((DefaultPort) port).getParent().getClass()
                        != HistoryCell.class
                && ((DefaultPort) port).getParent().getClass()
                        != DeepHistoryCell.class
                && ((DefaultPort) port).getParent()
                        != statechartRootCell;
```

```java
    }

    /**
     * Checks if a transition's target is ok.
     * @param edge is not interpreted.
     * @param port checks for the parent of this port, if it accepts
     incoming
     * transitions.
     * @return false if the port belongs to a initialState cell, a suspend
     cell
     * or the root cell.
     * @see org.jgraph.graph.GraphModel#acceptsTarget(java.lang.Object,
     *      java.lang.Object)
     */
    public boolean acceptsTarget(final Object edge, final Object port) {
        return port != null
                && ((DefaultPort) port).getParent().getClass()
                        != HistoryCell.class
                && ((DefaultPort) port).getParent().getClass()
                        != DeepHistoryCell.class
                && ((DefaultPort) port).getParent().getClass()
                        != InitialStateCell.class
                && ((DefaultPort) port).getParent().getClass()
                        != SuspendCell.class
                && ((DefaultPort) port).getParent()
                        != statechartRootCell;
    }

    /**
     * Constructs a graph model with a root graph cell and KIEL
     datastructure
     * with a root node.
     * @param modelSource Estudio or Matlab.
     * @param aGraphLayoutCache .
     */
    public MyGraphModel(final String modelSource,
        final GraphLayoutCache aGraphLayoutCache) {
        graphLayoutCache = aGraphLayoutCache;
        kielView = new View();
        kielStatechart = new StateChart();
        kielStatechart.setModelSource(modelSource);
        kielStatechart.setModelVersion("5.0");
        ORState root = new ORState(
            ResourceBundle.getString("newStatechart"));
        NodeLayoutInformation r = new CompositeStateLayoutInformation();
        r.setHeight(Preferences.getDefaultRootState().height);
        r.setWidth(Preferences.getDefaultRootState().width);
        r.setUpperLeftPoint(new kiel.graphicalInformations.Point(0, 0));
        kielView.setLayoutInformation(root, r);
        kielStatechart.setRootNode(root);
        statechartRootCell = add(kielStatechart.getRootNode());
    }

    /**
     * Constructs a model transforming the given KIEL datastructure.
     * @param aStatechart The statechart to be transformed in JGraph
     structure.
     * @param aView The related view of this statechart.
     * @param aGraphLayoutCache .
     */
    public MyGraphModel(final StateChart aStatechart, final View aView,
        final GraphLayoutCache aGraphLayoutCache) {
        graphLayoutCache = aGraphLayoutCache;
        kielView = aView;
        kielStatechart = aStatechart;
        statechartRootCell = add(aStatechart.getRootNode());
    }

    /**
     * Mapping Node --> NodeCell.
     * Creates a GraphCell object for the KIEL node object, for example
     * returns an ANDStateCell for an ANDState. The order of the if clauses
     is
     * significant. This method uses instanceof in order to accept new kiel
     * datastructure types.
     * @param graphicalObject The object for which a JGraph object has to be
     * created.
     * @return A new JGraph GraphCell object.
     */
    private DefaultGraphCell createGraphCellFor(
        final GraphicalObject graphicalObject) {
        DefaultGraphCell cell = null;
        String name = null;
        if (graphicalObject instanceof Node) {
            name = ((Node) graphicalObject).getName();
        }
        if (graphicalObject instanceof Choice) {
            cell = new ChoiceCell(name);
        } else if (graphicalObject instanceof DeepHistory) {
            cell = new DeepHistoryCell(name);
        } else if (graphicalObject instanceof DynamicChoice) {
            cell = new DynamicChoiceCell(name);
        } else if (graphicalObject instanceof ForkConnector) {
            cell = new ForkConnectorCell(name);
        } else if (graphicalObject instanceof History) {
            cell = new HistoryCell(name);
        } else if (graphicalObject instanceof InitialState) {
            cell = new InitialStateCell(name);
        } else if (graphicalObject instanceof Join) {
            cell = new JoinCell(name);
        } else if (graphicalObject instanceof Junction) {
            cell = new JunctionCell(name);
        } else if (graphicalObject instanceof Suspend) {
            cell = new SuspendCell(name);
        } else if (graphicalObject instanceof SynchState) {
            cell = new SynchStateCell(name);
        } else if (graphicalObject instanceof PseudoState) {
            cell = new PseudoStateCell(name);
```

```
        } else if (graphicalObject instanceof FinalANDState) {
            cell = new FinalANDStateCell(name);
        } else if (graphicalObject instanceof ANDState) {
            cell = new ANDStateCell(name);
        } else if (graphicalObject instanceof FinalORState) {
            cell = new FinalORStateCell(name);
        } else if (graphicalObject instanceof ORState) {
            cell = new ORStateCell(name);
        } else if (graphicalObject instanceof Region) {
            cell = new RegionCell(name);
        } else if (graphicalObject instanceof CompositeState) {
            cell = new CompositeStateCell(name);
        } else if (graphicalObject instanceof FinalState) {
            cell = new FinalStateCell(name);
        } else if (graphicalObject instanceof FinalSimpleState) {
            cell = new FinalSimpleStateCell(name);
        } else if (graphicalObject instanceof SimpleState) {
            cell = new SimpleStateCell(name);
        } else if (graphicalObject instanceof State) {
            cell = new StateCell(name);
        } else if (graphicalObject instanceof Node) {
            cell = new NodeCell(name);
        } else if (graphicalObject instanceof ConditionalTransition) {
            cell = new ConditionalTransitionCell();
        } else if (graphicalObject instanceof InitialArc) {
            cell = new InitialArcCell(null);
        } else if (graphicalObject instanceof NormalTermination) {
            cell = new NormalTerminationCell();
        } else if (graphicalObject instanceof StrongAbortion) {
            cell = new StrongAbortionCell();
        } else if (graphicalObject instanceof Suspension) {
            cell = new SuspensionCell();
        } else if (graphicalObject instanceof WeakAbortion) {
            cell = new WeakAbortionCell();
        } else if (graphicalObject instanceof Transition) {
            cell = new TransitionCell();
        } else if (graphicalObject instanceof DelimiterLine) {
            cell = new DelimiterLineCell(null);
        } else {
            logFile.log(2, "Unexpected type: " + graphicalObject.getClass())
      ;
        }
        nodesToCells.put(graphicalObject, cell);
        return cell;
    }

    /**
     * Adds a node to the model, creates a NodeCell and in case of an
     ANDState,
     * it calls addRegionOrORState(), in case of an ORState, it calls
     * addRegionOrORState() for its regions.
     * @param node The node to be added.
     * @return The created GraphCell for this node.
     */
    private NodeCell add(final Node node) {
        NodeCell cell = (NodeCell) createGraphCellFor(node);
        MyGraphModelUtilities.updateKielParentChildRelation(node);
        if (node instanceof ORState) {
            return addRegionOrORState((ORState) node, (ORStateCell) cell);
        }
        nodesToCells.put(node, cell);
        MyGraphConstants.setNodeAttributes(cell.getAttributes(), node,
            kielView);
        ParentMap parentMap = new ParentMap();
        if (!(node instanceof Region) && !(node instanceof ORState)) {
            parentMap.addEntry(new DefaultPort(), cell);
        }
        if (node instanceof ANDState) {
            Region[] regions = (Region[]) ((ANDState) node).getSubnodes()
                .toArray(new Region[0]);
            for (int i = 0; i < regions.length; i++) {
                parentMap.addEntry(addRegionOrORState(regions[i],
                    new RegionCell(regions[i].getName())), cell);
            }
            DelimiterLine[] delimiterLines = (DelimiterLine[]) ((ANDState)
                node).getDelimiterLines().toArray(new DelimiterLine[0]);
            for (int i = 0; i < delimiterLines.length; i++) {
                parentMap.addEntry(
                    add(delimiterLines[i], (ANDState) node), cell);
            }
        }
        insert(new Object[]{cell}, null, null, parentMap, null);
        return cell;
    }

    /**
     * Adds a delimiter line to the model, creates a DelimiterLineCell.
     * @param delimiterLine The delimiter line to be added.
     * @param parent the delimiter line's parent.
     * @return The created GraphCell for this delimiter line.
     */
    private DelimiterLineCell add(final DelimiterLine delimiterLine,
        final ANDState parent) {
        DelimiterLineCell cell =
            (DelimiterLineCell) createGraphCellFor(delimiterLine);
        nodesToCells.put(delimiterLine, cell);
        MyGraphConstants.setGraphicalObject(cell.getAttributes(),
            delimiterLine);
        DelimiterLineLayoutInformation layout =
            kielView.getLayoutInformation(delimiterLine);
        if (layout != null) {
            NodeLayoutInformation parentLayout =
                kielView.getLayoutInformation(parent);

            Point2D offset =
                MyGraphModelUtilities.calculateOffset(parent, kielView);
            offset.setLocation(
                offset.getX() + parentLayout.getXPos(),
```

```
                offset.getY() + parentLayout.getYPos());
            Rectangle2D bounds = new AttributeMap.SerializableRectangle2D(
                layout.getStart().getX() + offset.getX(),
                layout.getStart().getY() + offset.getY(),
                layout.getEnd().getX() - layout.getStart().getX(),
                layout.getEnd().getY() - layout.getStart().getY());
            GraphConstants.setBounds(cell.getAttributes(), bounds);
            if (bounds.getWidth() == 0) {
                bounds.setRect(bounds.getX(), bounds.getY(), 1,
                    bounds.getHeight());
            }
            if (bounds.getHeight() == 0) {
                bounds.setRect(bounds.getX(), bounds.getY(), bounds.getWidth
     (),
                    1);
            }
        }
        insert(new Object[]{cell}, null, null, null, null);
        return cell;
    }

    /**
     * Adds an ORState or Region to the model and connects this node with
     its
     * children (recursively!).
     * @param node The node to be added.
     * @param cell The corresponding GraphCell.
     * @return the cell parameter.
     */
    private NodeCell addRegionOrORState(final CompositeState node,
        final CompositeStateCell cell) {

        nodesToCells.put(node, cell);
        MyGraphConstants.setNodeAttributes(cell.getAttributes(), node,
            kielView);
        Node[] childNodes = null;
        if (node instanceof CompositeState) {
            childNodes = (Node[]) ((CompositeState) node).getSubnodes()
                .toArray(new Node[0]);
        }
        MyGraphModelUtilities.updateKielParentChildRelation(node);
        Map transportMap = new HashMap();
        ConnectionSet cs = new ConnectionSet();
        ParentMap parentMap = new ParentMap();
        if (!(node instanceof Region)) {
            parentMap.addEntry(new DefaultPort(), cell);
        }
        if (childNodes != null) {
            for (int i = 0; i < childNodes.length; i++) {
                parentMap.addEntry(add(childNodes[i]), cell);
            }
            // (for insert)
            for (int i = 0; i < childNodes.length; i++) {
                Node child = (Node) childNodes[i];
                Transition[] outgoingTransitions = (Transition[]) child
                    .getOutgoingTransitions().toArray(new Transition[0]);
                for (int j = 0; j < outgoingTransitions.length; j++) {
                    TransitionCell transitionCell = (TransitionCell)
                        createGraphCellFor(outgoingTransitions[j]);
                    AttributeMap newAttributes = new AttributeMap();
                    cs.connect(transitionCell,
                        getGraphCell(child).getChildAt(0),
                        getGraphCell(outgoingTransitions[j].getTarget())
                            .getChildAt(0));
                    parentMap.addEntry(transitionCell, cell);
                    MyGraphConstants.setTransitionAttributes(newAttributes,
                        outgoingTransitions[j]);
                    // Set Line Style...
                    EdgeLayoutInformation layout =
                        kielView.getLayoutInformation(outgoingTransitions[j
     ]);

                    if (layout.getAllPathElements() != null
                        && layout.getAllPathElements().size() > 2
                        && layout.getAllPathElements().get(1)
                            instanceof LinetoPath) {
                        GraphConstants.setLineStyle(
                            newAttributes, GraphConstants.STYLE_ORTHOGONAL);
                    }
                    GraphConstants.setPoints(newAttributes,
                        MyGraphModelUtilities
                            .convertTransitionPointsKielToJGraph(
                                outgoingTransitions[j], kielView));
                    transportMap.put(transitionCell, newAttributes);
                }
            }
        }
        insert(new Object[]{cell}, transportMap, cs, parentMap, null);
        return cell;
    }

    /**
     * Is called by EditorModeAddNode.doPlaceAction(). Creates
     * a KIEL datastructure object and fills the KIEL view with the current
     * mouse coordinates and the initialDimension for the nodeClass
     * @param nodeClass Which kind of node has to be created?
     * @param x The x-coordinate for the new node.
     * @param y The y-coordinate for the new node.
     * @param aGraphLayoutCache .
     */
    public void addNode(final Class nodeClass, final double x, final double
     y,
        final GraphLayoutCache aGraphLayoutCache) {

        // 1. Create a KIEL-node object...
        Node node = MyGraphModelUtilities.createKIELNodeFor(nodeClass);

        NodeLayoutInformation r = null;
```

```java
        if (node instanceof CompositeState) {
            r = new CompositeStateLayoutInformation();
        } else {
            r = new NodeLayoutInformation();
        }
        r.setUpperLeftPoint(new kiel.graphicalInformations.Point(
            (int) x, (int) y));
        r.setHeight(MyGraphModelUtilities.getInitialDimensionOf(nodeClass)
            .height);
        r.setWidth(MyGraphModelUtilities.getInitialDimensionOf(nodeClass)
            .width);
        kielView.setLayoutInformation(node, r);

        // 2. Transform to JGraph-nodeCell object...
        NodeCell cell = add(node);
        changeParent(cell, aGraphLayoutCache);
    }

    /**
     * Is called by EditorMode.isValidPosition(). Checks if a node of a
     * particular type can be added to the graph at the given position.
     * @param aCellClass which type of kielStatechart element should be
     * tested to create?
     * @param p at which point should the kielStatechart element be placed?
     * @param checkedCellClasses which are the generally accepted parent
     * classes? (ORState and Region for InitialStates for example)
     * @param excludeRootCell Is the kielStatechart root cell generally an
     * acceptable place for the new element?
     * @return true if can be added.
     */
    public boolean canBeAdded(final Class aCellClass, final Point2D p,
            final Class[] checkedCellClasses, final boolean excludeRootCell)
      {

        Object cell = getSmallestCellOfType(GraphCell.class,
            new Rectangle2D.Double(p.getX(), p.getY(), 1, 1), false);
        if (cell == null
                || !(cell instanceof CompositeStateCell)
                || MyGraphConstants.isCollapsed(
                    ((DefaultGraphCell) cell).getAttributes())
                || (excludeRootCell && cell == statechartRootCell)
                || !Arrays.asList(checkedCellClasses).contains(
                        cell.getClass())) {
            return false;
        } else {
            List children = ((CompositeStateCell) cell).getChildren();
            boolean existsAlready = false;
            for (int i = 0; !existsAlready && i < children.size(); i++) {
                if (children.get(i).getClass() == aCellClass) {
                    existsAlready = true;
                }
            }
            return !existsAlready;
        }
    }

    /**
     * Checks if a node of the given class would intersect any other node if
     * the node is placed at the given coordinates.
     * @param nodeClass which type of node?
     * @param x placing position
     * @param y placing position
     * @return true if intersects
     */
    public boolean intersect(final Class nodeClass, final double x,
        final double y) {

        Dimension d = MyGraphModelUtilities.getInitialDimensionOf(nodeClass)
    ;
        Rectangle2D initialBounds =
            new Rectangle2D.Double(x, y, d.width, d.height);
        return intersect(null, initialBounds, true, null, false);
    }

    /**
     * Checks if all children of the given graph cell are inside of a given
     * rectangle. This method can be used to check, if a resizing of a
    parent
     * state is allowed.
     * @param parent the cell whose children are to be checked.
     * @param parentsNewBounds Are these children in these new bounds?
     * @return true if all children are spatially inside its parent.
     */
    public boolean areAllChildrenInside(final DefaultGraphCell parent,
        final Rectangle2D parentsNewBounds) {

        for (int i = 0; i < parent.getChildCount(); i++) {
            if (!parentsNewBounds.contains(
                        MyGraphModelUtilities.getBoundsOf(parent.getChildAt(
    i)
                            , false))
                    && !(parent.getChildAt(i) instanceof Port)
                    && !(parent.getChildAt(i) instanceof DelimiterLineCell)
                    && !(parent.getChildAt(i) instanceof TransitionCell)) {
                // TODO Take care of the ports!?

                if (parent instanceof ANDStateCell) {
                    for (int j = 0; j < parent.getChildCount(); j++) {
                        if (!parentsNewBounds.intersects(
                                    MyGraphModelUtilities.getBoundsOf(
                                        parent.getChildAt(j), false))
                                && !(parent.getChildAt(j)
                                    instanceof Port)
                                && !(parent.getChildAt(j)
                                    instanceof DelimiterLineCell)
                                && !(parent.getChildAt(j)
                                    instanceof TransitionCell)) {
                            // TODO Take care of the ports!?
```

```
                        return false;
                    }
                }
                for (int j = 0; j < parent.getChildCount(); j++) {
                    boolean b = areAllChildrenInside(
                        (DefaultGraphCell) parent.getChildAt(j),
                        parentsNewBounds);
                    if (!b) {
                        return false;
                    }
                }
            } else {
                return false;
            }
        }
    }
    return true;
}

/**
 * Returns all GraphCells of the GraphModel starting at the given root
 cell.
 * @param rootCell Starting point in the hierarchy.
 * @return All cells below the given root cell.
 */
private DefaultGraphCell[] getAllCells(final Object rootCell) {
    return (DefaultGraphCell[]) getDescendants(this,
        new Object[] {rootCell}).toArray(new DefaultGraphCell[0]);
}

/**
 * Checks whether the given testBounds intersect one of the graph cell
 * bounds except for the excluded cell (and except
 * the children of the excluded cell).
 * @param excludedCell the cell which is dragged
 * @param testBounds the cell which is moving or the preview of the
 * dragged cell
 * @param excludeChildrenOfExcludedCell .
 * @param anExcludedCells .
 * @param orContain if true, then return true if intersect OR contain
 * @return true if intersect
 */
public boolean intersect(final GraphCell excludedCell,
    final Rectangle2D testBounds,
    final boolean excludeChildrenOfExcludedCell,
    final List anExcludedCells, final boolean orContain) {

    DefaultGraphCell[] all = getAllCells();
    List allExcludedCells = new ArrayList();
    if (excludeChildrenOfExcludedCell) {
        allExcludedCells.addAll(Arrays.asList(getAllCells(excludedCell))
 );
    } else if (excludedCell != null) {
        allExcludedCells.add(excludedCell);
    }
    if (anExcludedCells != null) {
        allExcludedCells.addAll(anExcludedCells);
    }
    for (int i = 0; i < all.length; i++) {
        if (!allExcludedCells.contains(all[i])
                && !(all[i] instanceof TransitionCell)
                && !(all[i] instanceof DelimiterLineCell)
                && !(all[i] instanceof Port)
                && !isInvisible(all[i])) {
            if (all[i] == null) {
                logFile.log(2, "kielGraphModel.intersect Null-Value");
            } else if (intersect(all[i], testBounds, orContain)
                && !(excludedCell instanceof ANDStateCell
                        && all[i] instanceof RegionCell)
                && !(excludedCell instanceof DelimiterLineCell
                        && all[i] instanceof RegionCell)
                && !(excludedCell instanceof DelimiterLineCell
                        && all[i] instanceof ANDStateCell)
                ) {
                        // Resize ANDState
                return true;
            }
        }
    }
    return false;
}

/**
 * .
 * @param cell .
 * @return .
 */
private boolean isInvisible(final DefaultGraphCell cell) {
    DefaultGraphCell parent = (DefaultGraphCell) cell.getParent();
    while (parent != null) {
        if (MyGraphConstants.isCollapsed(parent.getAttributes())) {
            return true;
        }
        parent = (DefaultGraphCell) parent.getParent();
    }
    return false;
}

/**
 * Checks if aCell's bounds intersect bBounds.
 * @param aCell .
 * @param bBounds .
 * @param orBBoundsContainsACell if true, then return true if
 * intersect or (bBounds contain aCell)
 * @return true if intersects.
 */
public boolean intersect(final DefaultGraphCell aCell,
    final Rectangle2D bBounds, final boolean orBBoundsContainsACell) {
```

```
    Rectangle2D aBounds = MyGraphModelUtilities
        .getBoundsOf(aCell, false).getBounds2D();
    aBounds.setRect(
        aBounds.getX() - Preferences.getIntersectionSensitivity(),
        aBounds.getY() - Preferences.getIntersectionSensitivity(),
        aBounds.getWidth() + 2 * Preferences.getIntersectionSensitivity
 (),
        aBounds.getHeight() + 2 * Preferences.getIntersectionSensitivity
 ());

    if (aBounds == null) {
        logFile.log(2, "aBounds is null");
        return true;
    }
    if (bBounds == null) {
        logFile.log(2, "bBounds is null");
        return true;
    }
    Rectangle2D theBigsValidArea = null;
    Rectangle2D theSmall = null;
    Rectangle2D theBig = null;

    if (aBounds.getWidth() < bBounds.getWidth()) {
        theBigsValidArea = bBounds;
        theSmall = aBounds;
        theBig = bBounds;
    } else {
        theBigsValidArea = MyGraphModelUtilities.getValidArea(aCell);
        theSmall = bBounds;
        theBig = aBounds;
    }
    final boolean onlyIntersects =
        (!theBig.contains(theSmall) && theBig.intersects(theSmall))
            || (!theBigsValidArea.contains(theSmall)
                && theBigsValidArea.intersects(theSmall))
            || (theBig.contains(theSmall)
                && !theBigsValidArea.contains(theSmall));
    if (!orBBoundsContainsACell) {
        return onlyIntersects;
    } else {
        return onlyIntersects || bBounds.contains(aBounds);
    }
}

/**
 * Finds out the smallest element which contains the given cell's bounds
 * completely. This is used whenever the given cell has been moved and
 you
 * want to assign to it the new spatial parent.
 * @param child .
 * @param aGraphLayoutCache .
 */
public void changeParent(final DefaultGraphCell child,
    final GraphLayoutCache aGraphLayoutCache) {

    DefaultGraphCell cell = child;
    if (cell instanceof TransitionCell) {
        cell = (DefaultGraphCell) ((DefaultPort) ((TransitionCell) cell)
            .getSource()).getParent();
    }
    Rectangle2D cellBounds =
        GraphConstants.getBounds(cell.getAttributes()).getBounds2D();
    // consider the case that 2 states with same bounds overlap each
 other
    cellBounds.setRect(cellBounds.getX() - 1, cellBounds.getY() - 1,
        cellBounds.getWidth() + 1, cellBounds.getHeight() + 1);
    CellView newSmallestParent =
        MyGraphModelUtilities.getSmallestViewOfType(
        CompositeStateView.class, cellBounds, aGraphLayoutCache, true,
        cell);
    if (newSmallestParent != null
        && newSmallestParent.getCell() /*NEW*/
            != ((DefaultGraphCell) child).getParent()) {

        ParentMap parentMap = new ParentMap();
        parentMap.addEntry(child, newSmallestParent.getCell());
        edit(null, null, parentMap, null);
        // setParent...
        updateKielParent(child);
    }
}

/**
 * Sets the cell's parent according to its attached KIEL node's parent.
 * @param cell .
 */
private void updateKielParent(final GraphCell cell) {
    if (cell instanceof NodeCell) {
        NodeCell nodeCell = (NodeCell) cell;
        Node node = (Node) MyGraphConstants.getGraphicalObject(
            nodeCell.getAttributes());
        if (getRootCell() == cell) {
            node.setParent(null);
        } else {
            node.setParent((CompositeState)
                MyGraphConstants.getGraphicalObject(((DefaultGraphCell)
                    nodeCell.getParent()).getAttributes()));
        }
    }
}

/**
 * Copies all attributes, the parent map and connection set of fromCell
 to
 * toCell.
 * @param fromCell .
 * @param toCellClass .
```

```
 * @return .
 */
public DefaultGraphCell convert(final DefaultGraphCell fromCell,
    final Class toCellClass) {
    DefaultGraphCell toCell = null;
    try {
        toCell = (DefaultGraphCell) toCellClass.getConstructor(
            new Class[] {Object.class}).newInstance(new Object[1]);
    } catch (InstantiationException e) {
        e.printStackTrace();
    } catch (IllegalAccessException e) {
        e.printStackTrace();
    } catch (InvocationTargetException e) {
        e.printStackTrace();
    } catch (NoSuchMethodException e) {
        e.printStackTrace();
    }
    getCurrentStateChart(); // Update the cells bounds...
    // Store the current fromCell KIEL data structure
    GraphicalObject fromGraphicalObject =
        MyGraphConstants.getGraphicalObject(fromCell.getAttributes());
    Transition[] incomingTransitions = new Transition[0];
    Transition[] outgoingTransitions = new Transition[0];
    if (fromGraphicalObject instanceof Node) {
        incomingTransitions = (Transition[]) ((Node) fromGraphicalObject
 )
                .getIncomingTransitions().toArray(new Transition[0]);
        outgoingTransitions = (Transition[]) ((Node) fromGraphicalObject
 )
                .getOutgoingTransitions().toArray(new Transition[0]);
    }
    // ********** 1. UPDATE THE GRAPH CELL... **********
    Map transportMap = new HashMap();
    ConnectionSet cs = new ConnectionSet();
    ParentMap parentMap = new ParentMap();
    parentMap.addEntry(fromCell, null);
    parentMap.addEntry(toCell, fromCell.getParent());
    transportMap.put(toCell, fromCell.getAttributes().clone());
    toCell.setUserObject(fromCell.getUserObject()); //TODO Necessary?

    for (int i = 0; i < fromCell.getChildCount(); i++) {
        if (fromCell.getChildAt(i) instanceof Port) {

            DefaultPort fromCellPort = (DefaultPort) fromCell.getChildAt
 (i);
            DefaultPort toCellPort = new DefaultPort();

            Edge[] edges =
                (Edge[]) fromCellPort.getEdges().toArray(new Edge[0]);
            for (int eCount = 0; eCount < edges.length; eCount++) {
                if (edges[eCount].getSource() == fromCell.getChildAt(i)
                    && edges[eCount].getTarget() == fromCell.getChildAt(i))
 {
                    cs.disconnect(edges[eCount]);
                    cs.connect(
                        edges[eCount],
                        toCellPort,
                        toCellPort);
                } else if (edges[eCount].getSource()
                        == fromCell.getChildAt(i)) {

                    cs.disconnect(edges[eCount], true);
                    cs.connect(
                        edges[eCount],
                        toCellPort,
                        edges[eCount].getTarget());
                } else if (edges[eCount].getTarget()
                        == fromCell.getChildAt(i)) {

                    cs.disconnect(edges[eCount], false);
                    cs.connect(
                        edges[eCount],
                        edges[eCount].getSource(),
                        toCellPort);
                }
                parentMap.addEntry(edges[eCount], fromCell.getParent());
            }
            parentMap.addEntry(fromCellPort, null);
            parentMap.addEntry(toCellPort, toCell);
        } else {
            parentMap.addEntry(fromCell.getChildAt(i), toCell);
        }
    }
    if (toCell instanceof TransitionCell) {
        cs.connect(
            toCell,
            ((TransitionCell) fromCell).getSource(),
            ((TransitionCell) fromCell).getTarget());
    }
    remove(new Object[]{fromCell});
    insert(new Object[]{toCell}, transportMap, null, parentMap, null);
    insert(cs.getChangedEdges().toArray(), null, cs, null, null);

    GraphicalObject toGraphicalObject =
        createKIELElementFor(toCell, (GraphCell) toCell.getParent());
    // ********** 2. UPDATE THE GRAPHICAL OBJECT... **********
    if (toCell instanceof NodeCell) {

        Node fromNode = (Node) fromGraphicalObject;
        Node toNode = (Node) toGraphicalObject;

        // A. Parent-Child relation...

        ((CompositeState) fromNode.getParent()).removeSubnode(fromNode);
        ((CompositeState) fromNode.getParent()).addSubnode(toNode);

        toNode.setParent(fromNode.getParent());
```

```
            // B. Variables and Events...
            if (fromNode instanceof State && toNode instanceof State) {
                ((State) toNode).setDoActivity(
                    ((State) fromNode).getDoActivity());
                ((State) toNode).setEntry(
                    ((State) fromNode).getEntry());
                ((State) toNode).setExit(
                    ((State) fromNode).getExit());
            }
            if (fromNode instanceof CompositeState
                && toNode instanceof CompositeState) {

                ((CompositeState) toNode).addLocalEvents(
                    ((CompositeState) fromNode).getLocalEvents());
                ((CompositeState) toNode).addVariables(
                    ((CompositeState) fromNode).getVariables());
            }

            // C. Incoming Transitions...
            for (int i = 0; i < incomingTransitions.length; i++) {
                toNode.getIncomingTransitions().remove(incomingTransitions[i
    ]);
                incomingTransitions[i].setTarget(toNode);
                incomingTransitions[i].getSource()
                    .getOutgoingTransitions().remove(incomingTransitions[i])
    ;
                incomingTransitions[i].setSource(
                    incomingTransitions[i].getSource());
            }

            // D. Outgoing Transitions...
            for (int i = 0; i < outgoingTransitions.length; i++) {
                toNode.getOutgoingTransitions().remove(outgoingTransitions[i
    ]);
                outgoingTransitions[i].setSource(toNode);
                outgoingTransitions[i].getTarget()
                    .getIncomingTransitions().remove(outgoingTransitions[i])
    ;
                outgoingTransitions[i].setTarget(
                    outgoingTransitions[i].getTarget());
            }
            Node[] children = (Node[]) // TODO necessary?
                kielStatechart.getRootNode().getSubnodes().toArray(new Node
    [0]);

            AttributeMap newAttributes = new AttributeMap();
            MyGraphConstants.setNodeAttributes(
                newAttributes, toNode, kielView);
            transportMap = new HashMap();
            transportMap.put(toCell, newAttributes);
            edit(transportMap, null, null, null);
        } else {
            Transition fromTransition = (Transition) fromGraphicalObject;
            Transition toTransition = (Transition) toGraphicalObject;
            // update the grpahicalObject...
            toTransition.setSource(fromTransition.getSource());
            toTransition.setTarget(fromTransition.getTarget());
            toTransition.setPriority(fromTransition.getPriority());
            toTransition.setLabel(fromTransition.getLabel());
//Prio-Fix    toTransition.getSource().addOutgoingTransition(toTransition);
//            toTransition.getTarget().addIncomingTransition(toTransition);
            fromTransition.getSource().removeOutgoingTransition(
    fromTransition);
            fromTransition.getTarget().removeIncomingTransition(
    fromTransition);
            kielView.put(
                toTransition.getID(),
                kielView.getLayoutInformation(fromTransition));
            AttributeMap newAttributes = new AttributeMap();
            MyGraphConstants.setTransitionAttributes(
                newAttributes, toTransition);
            GraphConstants.setPoints(newAttributes,
                MyGraphModelUtilities.convertTransitionPointsKielToJGraph(
                    toTransition, kielView));
            transportMap = new HashMap();
            transportMap.put(toCell, newAttributes);
            edit(transportMap, null, null, null);
        }
        // @TODO What should be done here? Take care of undo support!
//      kielView.removeLayoutInformation(fromGraphicalObject);
        return toCell;
    }

    /**
     * .
     * @param isVerticalDelimiter .
     * @param bounds .
     * @param compositeStateUpperOffset .
     * @param start .
     * @param end .
     * @param regionCell1 .
     * @param regionCell2 .
     */
    private void setRegionBounds(final boolean isVerticalDelimiter,
        final Rectangle2D bounds, final int compositeStateUpperOffset,
        final Point2D start, final Point2D end,
        final RegionCell regionCell1, final RegionCell regionCell2) {
        if (isVerticalDelimiter) {
            final double leftNewWidth = start.getX() - bounds.getX();
            GraphConstants.setBounds(regionCell1.getAttributes(),
                new AttributeMap.SerializableRectangle2D(
                    bounds.getX(),
                    start.getY(),
                    leftNewWidth,
                    end.getY() - start.getY()));
            GraphConstants.setBounds(regionCell2.getAttributes(),
                new AttributeMap.SerializableRectangle2D(
```

```
                    start.getX(),
                    start.getY(),
                    bounds.getWidth() - leftNewWidth,
                    end.getY() - start.getY()));
        } else {
            final double upperNewHeight = start.getY() - bounds.getY()
                    - compositeStateUpperOffset;
            GraphConstants.setBounds(regionCell1.getAttributes(),
                new AttributeMap.SerializableRectangle2D(
                    bounds.getX(),
                    bounds.getY() + compositeStateUpperOffset,
                    bounds.getWidth(), upperNewHeight));
            GraphConstants.setBounds(regionCell2.getAttributes(),
                new AttributeMap.SerializableRectangle2D(
                    bounds.getX(), start.getY(), bounds.getWidth(),
                    bounds.getHeight() - upperNewHeight
                    - compositeStateUpperOffset));
        }
    }

    /**
     * First calculates which state will get the delimiter line, then
     converts
     * to an ANDState, if the calculated state is an ORState, and then
     either
     * the target state will get one new region or two. The children have to
     * copied to the new parents (=regions). (This is the most sophisticate
     and
     * complex method of MyGraphModel:-p)
     * @param start .
     * @param end .
     */
    public void addDelimiter(final Point2D start, final Point2D end) {
        // Which state gets a delimiter line (=at least one new region?)
        getCurrentStateChart(); // TODO avoid access to KIEL
        StateCell cell = (StateCell) getSmallestCellOfType(
            CompositeStateCell.class,
            new Rectangle2D.Double(start.getX(), start.getY(), 1, 1), false)
    ;

        if (cell == null || cell == statechartRootCell) {
            return;
        }
        GraphCell[] children =
            (GraphCell[]) cell.getChildren().toArray(new GraphCell[0]);
        ParentMap parentMap = new ParentMap();
        if (cell instanceof ORStateCell) {
//            Port oldPort = null;
//            for (int i = 0; i < cell.getChildCount(); i++) {
//                if (cell.getChildAt(i) instanceof Port) {
//                    oldPort = (Port) cell.getChildAt(i);
//                }
//            }
            cell = (StateCell) convert(cell, ANDStateCell.class);
//                parentMap.addEntry(oldPort, cell);
        }
        // now cell is a regionCell or an empty AndStateCell
        final boolean isVerticalDelimiter = start.getX() == end.getX();
        Rectangle2D bounds = GraphConstants.getBounds(cell.getAttributes());
        CompositeState compositeState = (CompositeState) MyGraphConstants
            .getGraphicalObject(cell.getAttributes());
        Rectangle2D compositeStateCellBounds = bounds;
//          if (compositeState instanceof Region) {
//              compositeState = compositeState.getParent();
//              compositeStateCellBounds = GraphConstants.getBounds(
//                  ((GraphCell) cell.getParent()).getAttributes());
//          }
        final int compositeStateUpperOffset =
            Properties.getUpperOffset(compositeState);
        // correct start and end point
        if (isVerticalDelimiter) {
            start.setLocation(start.getX(), compositeStateCellBounds.getY()
                + compositeStateUpperOffset);
            end.setLocation(end.getX(), bounds.getY() + bounds.getHeight());
        } else {
            start.setLocation(bounds.getX(), start.getY());
            end.setLocation(bounds.getX() + bounds.getWidth(), end.getY());
        }
        // create Regions...
        if (cell instanceof ANDStateCell) {
            // has no region yet and gets 2 new ones
            RegionCell regionCell1 = new RegionCell(null);
            RegionCell regionCell2 = new RegionCell(null);
            MyGraphConstants.setNodeAttributes(regionCell1.getAttributes(),
                (Region) createKIELElementFor(regionCell1, cell), kielView);
            MyGraphConstants.setNodeAttributes(regionCell2.getAttributes(),
                (Region) createKIELElementFor(regionCell2, cell), kielView);
            setRegionBounds(isVerticalDelimiter, bounds,
                compositeStateUpperOffset, start, end,
                regionCell1, regionCell2);
            // 1. Region
            ParentMap regionParentMap = new ParentMap();
            GraphCell[] childrenToMove = getBoundedChildren(children,
                    GraphConstants.getBounds(regionCell1.getAttributes()));
            for (int i = 0; i < childrenToMove.length; i++) {
                regionParentMap.addEntry(childrenToMove[i], regionCell1);
            }
            insert(
                new Object[]{regionCell1}, null, null, regionParentMap, null
);
            // 2. Region
            regionParentMap = new ParentMap();
            childrenToMove = getBoundedChildren(children, GraphConstants
                    .getBounds(regionCell2.getAttributes()));
            for (int i = 0; i < childrenToMove.length; i++) {
                regionParentMap.addEntry(childrenToMove[i], regionCell2);
            }
            insert(
```

```java
                new Object[]{regionCell2}, null, null, regionParentMap, null
        );
            parentMap.addEntry(regionCell1, cell);
            parentMap.addEntry(regionCell2, cell);
        } else {
            // the AndState gets 1 more Region
            RegionCell regionCell1 = (RegionCell) cell;
            RegionCell regionCell2 = new RegionCell(null);
            MyGraphConstants.setNodeAttributes(regionCell2.getAttributes(),
                (Region) createKIELElementFor(
                    regionCell2, (GraphCell) cell.getParent()), kielView);

            AttributeMap newRegion1Attributes = new AttributeMap();

            if (isVerticalDelimiter) {
                double leftNewWidth = start.getX() - bounds.getX();
                GraphConstants.setBounds(newRegion1Attributes,
                    new AttributeMap.SerializableRectangle2D(
                        bounds.getX(), start.getY(), leftNewWidth,
                        end.getY() - start.getY()));
                GraphConstants.setBounds(regionCell2.getAttributes(),
                    new AttributeMap.SerializableRectangle2D(
                        start.getX(), start.getY(),
                        bounds.getWidth() - leftNewWidth,
                        end.getY() - start.getY()));
            } else {
                double upperNewHeight = start.getY() - bounds.getY();
                GraphConstants.setBounds(newRegion1Attributes,
                    new AttributeMap.SerializableRectangle2D(
                        bounds.getX(), bounds.getY(), bounds.getWidth(),
                        upperNewHeight));

                GraphConstants.setBounds(regionCell2.getAttributes(),
                    new AttributeMap.SerializableRectangle2D(
                        bounds.getX(), start.getY(), bounds.getWidth(),
                        bounds.getHeight() - upperNewHeight));
            }

            Map transportMap = new HashMap();
            transportMap.put(regionCell1, newRegion1Attributes);
            edit(transportMap, null, null, null);
            ParentMap regionParentMap = new ParentMap();
            GraphCell[] childrenToMove = getBoundedChildren(children,
                    GraphConstants.getBounds(regionCell2.getAttributes()));
            for (int i = 0; i < childrenToMove.length; i++) {
                regionParentMap.addEntry(childrenToMove[i], regionCell2);
            }
            insert(new Object[]{regionCell2}, null, null, regionParentMap,
                null);
            cell = (StateCell) cell.getParent();
            parentMap.addEntry(regionCell2, cell);
        }
        DelimiterLine delimiterLine = new DelimiterLine();
        ANDState parent = (ANDState)
            MyGraphConstants.getGraphicalObject(cell.getAttributes());
        parent.addDelimiterLine(delimiterLine);
        DelimiterLineCell delimiterLineCell = add(delimiterLine, parent);
        GraphConstants.setBounds(delimiterLineCell.getAttributes(),
            new AttributeMap.SerializableRectangle2D(
                start.getX(), start.getY(), end.getX() - start.getX() + 1,
                end.getY() - start.getY() + 1));

        parentMap.addEntry(delimiterLineCell, cell);
        edit(null, null, parentMap, null);
        setDelimiterLineLayout(start, end, cell, delimiterLine,
            delimiterLineCell);
    }

    /**
     * Returns the subset of given children, that is contained in the given
     * rect.
     * Is called from within addDelimiter().
     * @param children .
     * @param rect .
     * @return .
     */
    private GraphCell[] getBoundedChildren(final GraphCell[] children,
        final Rectangle2D rect) {
        ArrayList list = new ArrayList();
        for (int i = 0; i < children.length; i++) {
            if (children[i] instanceof NodeCell
                && rect.contains(GraphConstants.getBounds(
                    ((GraphCell) children[i]).getAttributes()))) {

                list.add(children[i]);
                list.addAll(Arrays.asList(MyGraphModel.getEdges(
                    this, children[i], false)));
            }
        }
        return (GraphCell[]) list.toArray(new GraphCell[0]);
    }

    /**
     * Is just called from addDelimiter(..) and sets the LayoutInformation
     for
     * the new delimiter line. This code here was just extracted to fit the
     * maximum size of 150 lines.
     * @param start .
     * @param end .
     * @param cell .
     * @param delimiterLine .
     * @param delimiterLineCell .
     */
    private void setDelimiterLineLayout(final Point2D start, final Point2D
     end,
        final DefaultGraphCell cell, final DelimiterLine delimiterLine,
        final DelimiterLineCell delimiterLineCell) {
        // Delimiter lines have relative Coordinates to the *AND-State*...
```

```
        DelimiterLineLayoutInformation layout =
            new DelimiterLineLayoutInformation();
        Rectangle2D bounds = GraphConstants.getBounds(cell.getAttributes());
        start.setLocation(
            start.getX() - bounds.getX(), start.getY() - bounds.getY());
        end.setLocation(
            end.getX() - bounds.getX(), end.getY() - bounds.getY());
        layout.setStart(new Point((int) start.getX(), (int) start.getY()));
        layout.setEnd(new Point((int) end.getX(), (int) end.getY()));
        kielView.setLayoutInformation(delimiterLine, layout);
        MyGraphConstants.setGraphicalObject(delimiterLineCell.getAttributes
   (),
            delimiterLine);
    }

    /**
     * Helper method.
     * @param clip .
     * @return all cells that intersect the given rectangle.
     */
    public Object[] getAllCells(final Rectangle2D clip) {
        java.util.List result = new ArrayList();
        Object[] all = getAllCells();
        for (int i = 0; i < all.length; i++) {
            if (MyGraphModelUtilities.getBoundsOf(all[i], false)
                .intersects(clip)) {
                result.add(all[i]);
            }
        }
        return result.toArray();
    }

    /**
     * Returns the smallest node of the given cellClass, containing the
     given
     * bounds.
     * @param cellClass .
     * @param bounds .
     * @param useDefaultBoundsForCollapsedStates for every collapsed
     * composite state take its default bounds instead of its expanded
     bounds
     * @return only visible cells!
     */
    public Object getSmallestCellOfType(final Class cellClass,
        final Rectangle2D bounds,
        final boolean useDefaultBoundsForCollapsedStates) {

        final boolean flag = useDefaultBoundsForCollapsedStates; //as
     shortcut
        Object[] all = getAllCells(bounds);
        Object newSmallestParent = null;
        for (int i = 0; i < all.length; i++) {
            if (cellClass.isInstance(all[i])
                && !isInvisible((DefaultGraphCell) all[i])) {
                if (MyGraphModelUtilities.getBoundsOf(all[i], flag)
                    .contains(bounds)) {
                    if (newSmallestParent == null) {
                        newSmallestParent = all[i];
                    } else if (MyGraphModelUtilities.getBoundsOf(
                            newSmallestParent, flag).getWidth()
                            * MyGraphModelUtilities.getBoundsOf(
                                newSmallestParent, flag).getHeight()
                        > MyGraphModelUtilities.getBoundsOf(all[i], flag)
                            .getWidth()
                            * MyGraphModelUtilities.getBoundsOf(all[i], flag
   )
                                .getHeight()) {
                        newSmallestParent = all[i];
                    }
                }
            }
        }
        return newSmallestParent;
    }

    /**
     * Changes the line style of the given cell.
     * @param cell .
     * @param style .
     */
    public void changeLineStyle(final GraphCell cell, final int style) {

        Map map = new AttributeMap();
        GraphConstants.setLineStyle(map, style);
        Map transportMap = new Hashtable();
        transportMap.put(cell, map);
        edit(transportMap, null, null, null);
    }

    /**
     * Calls super.edit() and then calls changeKielNodeParent() just in case
     ,
     * that this needs to be updated.
     * @see org.jgraph.graph.GraphModel#edit(java.util.Map,
     *      org.jgraph.graph.ConnectionSet, org.jgraph.graph.ParentMap,
     *      javax.swing.undo.UndoableEdit[])
     */
    public synchronized void edit(final Map transportMap,
        final ConnectionSet cs, final ParentMap parentMap,
        final UndoableEdit[] edits) {
        super.edit(transportMap, cs, parentMap, edits);
        MyGraphModelUtilities.updateKielParent(parentMap);
    }

    /**
     * Calls super.insert() and then calls changeKielNodeParent() just in
     case,
     * that this needs to be updated.
```

```
 * @see org.jgraph.graph.GraphModel#insert(java.lang.Object[],
 *      java.util.Map, org.jgraph.graph.ConnectionSet,
 *      org.jgraph.graph.ParentMap, javax.swing.undo.UndoableEdit[])
 */
public void insert(final Object[] roots, final Map transportMap,
        final ConnectionSet cs, final ParentMap parentMap,
        final UndoableEdit[] edits) {
    super.insert(roots, transportMap, cs, parentMap, edits);
    MyGraphModelUtilities.updateKielParent(parentMap);
}

/**
 * removes a region as well and copies children from the removed region
 * (the lower right) to the to the upper left.
 * @param delimiterLineCell to be removed.
 */
private void removeDelimiterLine(
    final DelimiterLineCell delimiterLineCell) {

    ANDStateCell parentCell = (ANDStateCell) delimiterLineCell.getParent
();
    int regionCount = 0;

    Rectangle2D delimiterBounds =
        GraphConstants.getBounds(delimiterLineCell.getAttributes());

    final boolean isHorizontal = matches(delimiterBounds.getHeight(), 0)
;
    // Find the two corresponding regions...
    RegionCell upperLeftCell = null;
    RegionCell lowerRightCell = null;
    for (int j = 0; j < parentCell.getChildCount(); j++) {
        if (parentCell.getChildAt(j) instanceof RegionCell) {
            regionCount++;
            Rectangle2D regionBounds =
                GraphConstants.getBounds(
                    ((GraphCell) parentCell.getChildAt(j)).getAttributes
());
            if (isHorizontal
                // left corner
                && matches(regionBounds.getX(), delimiterBounds.getX())
                // length
                && matches(regionBounds.getWidth(),
                    delimiterBounds.getWidth())) {
                // region next to delimiterLine
                if (matches(regionBounds.getY()
                    + regionBounds.getHeight(),
                    delimiterBounds.getY())) {

                    upperLeftCell = (RegionCell) parentCell.getChildAt(j
);
                } else if (matches(delimiterBounds.getY(),
                    regionBounds.getY())) {

                    lowerRightCell = (RegionCell) parentCell.getChildAt(
j);
                }
            } else if (
                !isHorizontal
                // left corner
                && matches(regionBounds.getY(),
                    delimiterBounds.getY())
                // length
                && matches(regionBounds.getHeight(),
                    delimiterBounds.getHeight())) {
                // region next to delimiterLine
                if (matches(regionBounds.getX()
                    + regionBounds.getWidth(), delimiterBounds.getX()))
{

                    upperLeftCell = (RegionCell) parentCell.getChildAt(j
);
                } else if (matches(delimiterBounds.getX(),
                    regionBounds.getX())) {

                    lowerRightCell = (RegionCell) parentCell.getChildAt(
j);
                }
            }
        }
    }
    if (upperLeftCell == null) {
        logFile.log(0, "No upperLeft found");
        return;
    }
    if (lowerRightCell == null) {
        logFile.log(0, "No lowerRight found");
        return;
    }

    ParentMap pm = new ParentMap();
    // Copy from lower right to upper left...
    for (int i = 0; i < lowerRightCell.getChildCount(); i++) {
        if (lowerRightCell.getChildAt(i) instanceof NodeCell
            || lowerRightCell.getChildAt(i) instanceof TransitionCell) {
            pm.addEntry(lowerRightCell.getChildAt(i), upperLeftCell);
        }
    }
    Map transportMap = new HashMap();
    AttributeMap newAttributes = new AttributeMap();
    transportMap.put(upperLeftCell, newAttributes);
    Rectangle2D upperLeftBounds = GraphConstants.getBounds(
        upperLeftCell.getAttributes());
    Rectangle2D lowerRightBounds = GraphConstants.getBounds(
        lowerRightCell.getAttributes());
    if (isHorizontal) {
        GraphConstants.setBounds(newAttributes,
            new AttributeMap.SerializableRectangle2D(
```

```
                upperLeftBounds.getX(),
                upperLeftBounds.getY(),
                upperLeftBounds.getWidth(),
                upperLeftBounds.getHeight()
                + lowerRightBounds.getHeight()));
    } else {
        GraphConstants.setBounds(newAttributes,
            new AttributeMap.SerializableRectangle2D(
                upperLeftBounds.getX(),
                upperLeftBounds.getY(),
                upperLeftBounds.getWidth()
                + lowerRightBounds.getWidth(),
                upperLeftBounds.getHeight()));
    }
    edit(transportMap, null, pm, null);
    remove(new Object[] {lowerRightCell});

    if (regionCount == 2) {
        pm = new ParentMap();
        // Copy from the only region to the ANDState...
        for (int i = 0; i < upperLeftCell.getChildCount(); i++) {
            if (upperLeftCell.getChildAt(i) instanceof NodeCell
                || upperLeftCell.getChildAt(i) instanceof TransitionCell
 ) {
                pm.addEntry(upperLeftCell.getChildAt(i), parentCell);
            }
        }
        remove(new Object[] {upperLeftCell});
        edit(null, null, pm, null);
        convert(parentCell, ORStateCell.class);
    }
}

/**
 * Checks whether the two parameters are nearly equal.
 * @param a compare with b.
 * @param b compare with a.
 * @return true if a and b are nearly equal (+- 10)
 */
private static boolean matches(final double a, final double b) {
    final int tolerance = 15;
    return Math.abs(a - b) < tolerance;
}

/**
 * Calls super.remove() and then disconnects the given cells from the
 rest
 * of the tree and from the transitions.
 * @see org.jgraph.graph.GraphModel#remove(java.lang.Object[])
 */
public void remove(final Object[] roots) {
    if (roots == null) {
        return;
    }
    for (int i = 0; i < roots.length; i++) {
        GraphicalObject object = MyGraphConstants.getGraphicalObject(
            ((GraphCell) roots[i]).getAttributes());
        if (object instanceof DelimiterLine) {
            removeDelimiterLine((DelimiterLineCell) roots[i]);
        } else if (object instanceof Node) {
            // remove the node itself...
            if (((Node) object).getParent() instanceof CompositeState) {
                ((CompositeState) ((Node) object).getParent())
                    .removeSubnode((Node) object);
            }
            // Remove its transitions...
            Collection transitionCollection =
                ((Node) object).getIncomingTransitions();
            transitionCollection.addAll(
                ((Node) object).getOutgoingTransitions());
            Transition[] transitions =
                (Transition[]) transitionCollection.toArray(
                    new Transition[0]);
            for (int tCount = 0; tCount < transitions.length; tCount++)
 {
                remove(new Object[]{getGraphCell(transitions[tCount])});
            }
        } else if (object instanceof Transition) {
            ((Transition) object).getSource().removeOutgoingTransition(
                (Transition) object);
            ((Transition) object).getTarget().removeIncomingTransition(
                (Transition) object);
        }
    }
    super.remove(roots);
}

/**
 * Updates the KIEL datastructure and returns the current kielStatechart
 .
 * @return .
 */
public StateChart getCurrentStateChart() {
    // TODO Check why 2 times??
    updateKielObject(statechartRootCell);
    updateKielObject(statechartRootCell);
    return kielStatechart;
}

/**
 * Just returns the current KIEL view.
 * @return .
 */
public View getCurrentView() {
    return kielView;
}

/**
```

```
     * Updates the given cell's associated KIEL node. It updates the
     * parent-child relation, the name, the bounds / transition points, the
     * delimiter lines.
     * @param cell .
     * @return .
     */
    private GraphCell updateKielObject(final GraphCell cell) {

        if (cell instanceof NodeCell) {
            // Update parent-child relation...
            Node node = (Node) MyGraphConstants.getGraphicalObject(
                cell.getAttributes());
            updateKielParent(cell);

            // Update name...
            node.setName(cell.toString());

            // Update Bounds / Points
            NodeLayoutInformation layout = kielView.getLayoutInformation(
     node);
            Rectangle2D bounds = GraphConstants.getBounds(
                cell.getAttributes()).getBounds();
            if (bounds != null) {   // @TODO can be null!?
                // Clicking on some elements causes to hide them
                Rectangle2D expandedBounds = MyGraphConstants
                    .getExpandedBounds(cell.getAttributes());
                if (MyGraphConstants.isCollapsed(cell.getAttributes())
                    && expandedBounds != null) {
                    layout.setHeight((int) expandedBounds.getHeight());
                    layout.setWidth((int) expandedBounds.getWidth());
                } else {
                    layout.setHeight((int) bounds.getHeight());
                    layout.setWidth((int) bounds.getWidth());
                }
            }
            layout.setUpperLeftPoint(
                MyGraphModelUtilities.getRelativeCoordinates((NodeCell) cell
     ));

            // update children relation...
            if (node instanceof CompositeState) {
                ((CompositeState) node).getSubnodes().clear();
                if (node instanceof ANDState) {
                    ((ANDState) node).removeAllDelimiterLines();
                }
                if (MyGraphConstants.isCollapsed(cell.getAttributes())) {
                    expand(null, (CompositeStateCell) cell,
                        new Point2D.Double(0, 0), null, null, false);
                }
            }
            for (int i = 0; i < ((NodeCell) cell).getChildCount(); i++) {
                GraphCell childCell =
                    (GraphCell) ((NodeCell) cell).getChildAt(i);
                updateKielObject(childCell);
                if (node instanceof CompositeState
                    && childCell instanceof NodeCell) {

                    ((CompositeState) node).addSubnode(
                        (Node) MyGraphConstants.getGraphicalObject(
                            childCell.getAttributes()));
                } else if (node instanceof ANDState
                    && childCell instanceof DelimiterLineCell) {
                    ((ANDState) node).addDelimiterLine(
                        (DelimiterLine) MyGraphConstants.getGraphicalObject(
                            childCell.getAttributes()));
                }
            }
            if (layout instanceof CompositeStateLayoutInformation) {
                ((CompositeStateLayoutInformation) layout).setCollapsed(
                    MyGraphConstants.isCollapsed(cell.getAttributes()));
            }

            // Have KIEL-Transitions to be deleted?
            // 1. get the port...
            DefaultPort port = null;
            for (int cCount = 0; port == null
                && cCount < ((NodeCell) cell).getChildCount(); cCount++) {

                if (((NodeCell) cell).getChildAt(cCount)
                    instanceof DefaultPort) {
                    port = (DefaultPort) ((NodeCell) cell).getChildAt(cCount
     );
                }
            }
            Object[] outgoingTransitions = node.getOutgoingTransitions()
                .toArray();
            for (int i = 0; i < outgoingTransitions.length; i++) {
                // search for it...
                boolean found = false;
                for (int tCount = 0;
                    !found && tCount < port.getChildCount(); tCount++) {

                    if (MyGraphConstants.getGraphicalObject(
                        ((DefaultGraphCell) port.getChildAt(tCount))
                        .getAttributes()) == outgoingTransitions[i]) {
                        found = true;
                    }
                }
                if (!found) {
                    node.removeOutgoingTransition(
                        (kiel.dataStructure.Edge) outgoingTransitions[i]);
                }
            }
        } else if (cell instanceof TransitionCell) {

            Transition transition = (Transition) MyGraphConstants
                .getGraphicalObject(cell.getAttributes());
```

```
            // Update source and target...
            Node newSource = (Node) MyGraphConstants.getGraphicalObject(
                ((NodeCell) ((DefaultPort) ((TransitionCell) cell).getSource
        ())
                    .getParent()).getAttributes());
            transition.getSource().removeOutgoingTransition(transition);
            transition.setSource(newSource);
            Node newTarget = (Node) MyGraphConstants.getGraphicalObject(
                ((NodeCell) ((DefaultPort) ((TransitionCell) cell).getTarget
        ())
                    .getParent()).getAttributes());
            transition.getTarget().removeIncomingTransition(transition);
            transition.setTarget(newTarget);

            // Update the points...
            kielView.getLayoutInformation(transition).setPath(
                MyGraphModelUtilities.convertTransitionPointsJGraphToKiel(
                    (TransitionCell) cell, kielView, transition));

            // Update the Label position...
            updateKielTransitionLabelsPosition(
                (TransitionCell) cell, null);
        } else if (cell instanceof DelimiterLineCell) {

            DelimiterLine delimiterLine =
                (DelimiterLine) MyGraphConstants.getGraphicalObject(
                    cell.getAttributes());

            DelimiterLineLayoutInformation layout =
                kielView.getLayoutInformation(delimiterLine);
            Rectangle2D bounds =
                GraphConstants.getBounds(cell.getAttributes()).getBounds();
            Rectangle2D parentBounds =
                GraphConstants.getBounds(((DefaultGraphCell)
                ((DefaultGraphCell) cell).getParent()).getAttributes());

            layout.setStart(new Point(
                (int) (bounds.getX() - parentBounds.getX()),
                (int) (bounds.getY() - parentBounds.getY())));
                layout.setEnd(new Point(
                    (int) (bounds.getX() + bounds.getWidth()
                        - parentBounds.getX()),
                    (int) (bounds.getY() + bounds.getHeight()
                        - parentBounds.getY())));
        } else if (!(cell instanceof Port)) {
            logFile.log(2, "Unexpected cell type:" + cell.getClass());
        }
        return cell;
    }

    /**
     * Defines the mapping GraphCell --> GraphicalObject and creates the
     * appropriate GraphicalObject for the given cell.
     * @param cell .
     * @return .
     */
    private GraphicalObject createKIELElementFor(final GraphCell cell) {

        GraphicalObject graphicalObject = null;
        if (cell.getClass() == ANDStateCell.class) {
            graphicalObject = new ANDState();
        } else if (cell.getClass() == ChoiceCell.class) {
            graphicalObject = new Choice();
        } else if (cell.getClass() == ConditionalTransitionCell.class) {
            graphicalObject = new ConditionalTransition();
        } else if (cell.getClass() == DeepHistoryCell.class) {
            graphicalObject = new DeepHistory();
        } else if (cell.getClass() == DynamicChoiceCell.class) {
            graphicalObject = new DynamicChoice();
        } else if (cell.getClass() == FinalANDStateCell.class) {
            graphicalObject = new FinalANDState();
        } else if (cell.getClass() == FinalORStateCell.class) {
            graphicalObject = new FinalORState();
        } else if (cell.getClass() == FinalSimpleStateCell.class) {
            graphicalObject = new FinalSimpleState();
        } else if (cell.getClass() == FinalStateCell.class) {
            graphicalObject = new FinalState();
        } else if (cell.getClass() == ForkConnectorCell.class) {
            graphicalObject = new ForkConnector();
        } else if (cell.getClass() == HistoryCell.class) {
            graphicalObject = new History();
        } else if (cell.getClass() == InitialArcCell.class) {
            graphicalObject = new InitialArc();
        } else if (cell.getClass() == InitialStateCell.class) {
            graphicalObject = new InitialState();
        } else if (cell.getClass() == JoinCell.class) {
            graphicalObject = new Join();
        } else if (cell.getClass() == JunctionCell.class) {
            graphicalObject = new Junction();
        } else if (cell.getClass() == NormalTerminationCell.class) {
            graphicalObject = new NormalTermination();
        } else if (cell.getClass() == ORStateCell.class) {
            graphicalObject = new ORState();
        } else if (cell.getClass() == RegionCell.class) {
            graphicalObject = new Region();
        } else if (cell.getClass() == SimpleStateCell.class) {
            graphicalObject = new SimpleState();
        } else if (cell.getClass() == StateCell.class) {
            graphicalObject = new State();
        } else if (cell.getClass() == StrongAbortionCell.class) {
            graphicalObject = new StrongAbortion();
        } else if (cell.getClass() == SuspensionCell.class) {
            graphicalObject = new Suspension();
        } else if (cell.getClass() == StrongAbortionCell.class) {
            graphicalObject = new StrongAbortion();
        } else if (cell.getClass() == SynchStateCell.class) {
            graphicalObject = new SynchState();
        } else if (cell.getClass() == TransitionCell.class) {
```

```
            graphicalObject = new Transition();
        } else if (cell.getClass() == WeakAbortionCell.class) {
            graphicalObject = new WeakAbortion();
        } else if (cell.getClass() == DelimiterLineCell.class) {
            graphicalObject = new DelimiterLine();
        }
        nodesToCells.put(graphicalObject, cell);

        return graphicalObject;
    }

    /**
     * Calls createKIELElementFor(GraphCell) and initializes the new
     * GraphicalObject with the layout information (coordinates).
     * @param cell .
     * @param parentCell .
     * @return .
     */
    private GraphicalObject createKIELElementFor(final GraphCell cell,
        final GraphCell parentCell) {

        GraphicalObject graphicalObject = createKIELElementFor(cell);

        if (graphicalObject instanceof Transition) {
            kielView.setLayoutInformation(
                graphicalObject, new EdgeLayoutInformation());
        } else {
            NodeLayoutInformation layout = null;
            if (graphicalObject instanceof CompositeState) {
                layout = new CompositeStateLayoutInformation();
            } else {
                layout = new NodeLayoutInformation();
            }
            kielView.setLayoutInformation(graphicalObject, layout);
            ((Node) graphicalObject).setParent(
                (CompositeState) MyGraphConstants.getGraphicalObject(
                    parentCell.getAttributes()));
            Rectangle2D bounds = GraphConstants.getBounds(cell.getAttributes
 ());
            if (bounds != null) {
                layout.setWidth((int) bounds.getWidth());
                layout.setHeight((int) bounds.getHeight());
            }
            layout.setUpperLeftPoint(
                MyGraphModelUtilities.getRelativeCoordinates((NodeCell) cell
 ));
        }
        return graphicalObject;
    }

    /**
     * Creates a new KIEL-transition connecting the given ports according to
     * the line type and transition type currently choosen in the editor.
     * @param source .
     * @param target .
     * @param type .
     * @param line .
     * @param location .
     * @throws Exception If source or target are not valid.
     */
    public void addTransition(final Port source, final Port target,
        final int type, final int line, final Point2D location)
        throws Exception {

        if (((DefaultGraphCell) source).getParent().getParent()
            != ((DefaultGraphCell) target).getParent().getParent()) {
          throw new Exception(ResourceBundle.getString(
              "interlevelTransitionsNotAllowed"));
        } else if (!acceptsSource(null, source)) {
            throw new Exception(ResourceBundle.getString(
                "sourceNotAllowed"));
        } else if (!acceptsTarget(null, target)) {
            throw new Exception(ResourceBundle.getString(
                "targetNotAllowed"));
        } else if (((DefaultGraphCell) source).getParent()
            instanceof InitialStateCell
            && getEdges(this, ((DefaultGraphCell) source).getParent(),
            false).length > 0) {
            throw new Exception(ResourceBundle.getString(
                "hasAlreadyOutgoingTransition"));
        }
        // following algorithm relies on the KIEL structure
        // @TODO Do not use the KIEL structure!
        Node sourceNode = (Node) MyGraphConstants.getGraphicalObject(
            ((GraphCell) ((DefaultPort) source).getParent()).getAttributes()
 );
        Node targetNode = (Node) MyGraphConstants.getGraphicalObject(
            ((GraphCell) ((DefaultPort) target).getParent()).getAttributes()
 );

        // 1. Create a KIEL-transition object...
        Transition transition;
        if (sourceNode instanceof InitialState) {
            transition = new InitialArc();
        } else if (sourceNode instanceof DynamicChoice) {
            transition = new ConditionalTransition();
        } else if (sourceNode instanceof Suspend) {
            transition = new Suspension();
        } else if (type == 0) {
            transition = new NormalTermination();
        } else if (type == 1) {
            transition = new WeakAbortion();
        } else if (type == 2) {
            transition = new StrongAbortion();
        } else {
            transition = new Transition();
        }
        transition.setSource(sourceNode);
```

```java
        transition.setTarget(targetNode);
        transition.setPriority(0);
        transition.setPriority(
            MyGraphModelUtilities.getHighestPriority(sourceNode) + 1);

        kielView.setLayoutInformation(transition, new EdgeLayoutInformation
    ());

        // 2. Transform to JGraph-transitionCell...
        CompositeStateCell transitionCellParent = (CompositeStateCell)
            ((DefaultPort) source).getParent().getParent();

        TransitionCell transitionCell =
            (TransitionCell) createGraphCellFor(transition);

        AttributeMap newAttributes = new AttributeMap();
        MyGraphConstants.setTransitionAttributes(newAttributes,
            transition);

        if (source == target) {
            Rectangle2D sourceBounds = GraphConstants.getBounds(
                ((DefaultGraphCell) ((DefaultPort) source).getParent())
                    .getAttributes());
            final int loopX = 50;
            final int loopY = 25;
            List points = new ArrayList();
            points.add(new AttributeMap.SerializablePoint2D(
                sourceBounds.getX(), sourceBounds.getY()));
            points.add(new AttributeMap.SerializablePoint2D(
                sourceBounds.getX() + sourceBounds.getWidth() + loopX,
                sourceBounds.getY() + sourceBounds.getHeight() / 2 - loopY))
    ;
            points.add(new AttributeMap.SerializablePoint2D(
                sourceBounds.getX() + sourceBounds.getWidth() + loopX,
                sourceBounds.getY() + sourceBounds.getHeight() / 2 + loopY))
    ;
            points.add(new AttributeMap.SerializablePoint2D(
                sourceBounds.getX(), sourceBounds.getY()));
            GraphConstants.setPoints(newAttributes, points);
        }
        if (line == 0) {
            GraphConstants.setLineStyle(
                newAttributes, GraphConstants.STYLE_SPLINE);
        } else if (line == 1) {
            GraphConstants.setLineStyle(
                newAttributes, GraphConstants.STYLE_BEZIER);
        } else if (line == 2) {
            GraphConstants.setLineStyle(
                newAttributes, GraphConstants.STYLE_ORTHOGONAL);
        }

        Map transportMap = new HashMap();
        transportMap.put(transitionCell, newAttributes);
        ConnectionSet cs = new ConnectionSet(
            transitionCell, source, target);

        ParentMap parentMap = new ParentMap();
        parentMap.addEntry(transitionCell, transitionCellParent);

        insert(new Object[]{transitionCell}, transportMap, cs,
            parentMap, null);
    }

    /**
     * .
     * @param transitionCell .
     * @param value .
     */
    public void changeTransitionPriority(final TransitionCell transitionCell
     ,
        final int value) {

        Map transportMap = new HashMap();

        final Transition transition = (Transition) MyGraphConstants
            .getGraphicalObject(transitionCell.getAttributes());

        transition.getPriority().setValue(value);

        AttributeMap map = new AttributeMap();
        transportMap.put(transitionCell, map);

        GraphConstants.setExtraLabels(map,
            new Object[]{"<" + transition.getPriority().getValue() + ">"});

        ArrayList outgoingTransitions =
            (ArrayList) transition.getSource().getOutgoingTransitions();
        Collections.sort(outgoingTransitions, new Comparator() {
            public int compare(final Object o1, final Object o2) {

                int result = ((Transition) o1).getPriority().getValue()
                    - ((Transition) o2).getPriority().getValue();
                if (result == 0 && o1 != o2) {
                    if (o1 == transition) {
                        return -1;
                    } else if (o2 == transition) {
                        return 1;
                    }
                }
                return result;
            }
        });
        for (int i = 0; i < outgoingTransitions.size(); i++) {
            Transition aTransition = (Transition) outgoingTransitions.get(i)
     ;
            aTransition.getPriority().setValue(i + 1);

            AttributeMap aMap = new AttributeMap();
```

```
            transportMap.put(getGraphCell(aTransition), aMap);
            GraphConstants.setExtraLabels(aMap,
                new Object[]{"<" + aTransition.getPriority().getValue() +
     ">"});
        }
        edit(transportMap, null, null, null);
    }

    /**
     * Toggles the collapsed state of the given cell. An expanding or
     collapsing
     * is done through the MorphingLayouter. "Create space" mechanisms are
     not
     * implemented here but in the MorphingLayouter!
     * @param aGraph .
     * @param cell .
     * @param aGraphLayoutCache for the exact current bounds.
     * @param listener .
     */
    public void flipCollapsed(final MyJGraph aGraph,
        final CompositeStateCell cell, final GraphLayoutCache
     aGraphLayoutCache,
        final MorphingLayouter.MyListener listener) {

        if (!MyGraphConstants.isCollapsed(cell.getAttributes())) {

            Map transportMap = new HashMap();
            AttributeMap newAttributes = new AttributeMap();
            transportMap.put(cell, newAttributes);

            // change collapsed attribute...
            MyGraphConstants.setCollapsed(newAttributes,
                !MyGraphConstants.isCollapsed(cell.getAttributes()));

            // set expanded bounds attribute...
            Rectangle2D exBounds = new AttributeMap.SerializableRectangle2D
     ();
            exBounds.setRect(
                aGraphLayoutCache.getMapping(cell, false).getBounds());
            MyGraphConstants.setExpandedBounds(newAttributes, exBounds);

            edit(transportMap, null, null, null);
            //...
            Dimension d = MyGraphModelUtilities.getInitialDimensionOf(
                ORState.class);
            Rectangle2D newBounds = new Rectangle2D.Double(
                exBounds.getX(), exBounds.getY(), d.width, d.height);

            processIsCollapsed(cell, aGraph, false);

            new MorphingLayouter(aGraph, listener, true)
                .collapseCompositeState(cell, newBounds);

            aGraph.getOverviewGraph().getSelectionModel().clearSelection();
        } else {
            expand(aGraph, cell,
                new Point2D.Double(0, 0), aGraphLayoutCache, listener, true)
     ;
        }
    }

    /**
     * .
     * @param cell .
     * @param aGraph .
     * @param wholeStatechart .
     */
    public void processIsCollapsed(final CompositeStateCell cell,
        final MyJGraph aGraph,
        final boolean wholeStatechart) {

        hideContents(cell, aGraph);

        for (int i = 0; wholeStatechart && i < cell.getChildCount(); i++) {
            if (cell.getChildAt(i) instanceof CompositeStateCell) {
                processIsCollapsed(
                    (CompositeStateCell) cell.getChildAt(i), aGraph,
                    wholeStatechart);
            }
        }
    }

    /**
     * .
     * @param cell .
     * @param aGraph .
     */
    private void hideContents(final CompositeStateCell cell,
        final MyJGraph aGraph) {
        if (MyGraphConstants.isCollapsed(cell.getAttributes())) {

            aGraph.getGraphLayoutCache().collapse(
                new Object[]{cell});
            aGraph.getOverviewGraph().getGraphLayoutCache().collapse(
                new Object[]{cell});
        }
    }
    /**
     * Is called by the flipCollapsed() method recursively. "Create space"
     * mechanisms are not implemented here but in the MorphingLayouter!
     * @param aGraph .
     * @param cell .
     * @param defaultDelta .
     * @param aGraphLayoutCache .
     * @param listener .
     * @param doExpand .
     */
    private void expand(final MyJGraph aGraph,
```

```java
        final CompositeStateCell cell,
        final Point2D defaultDelta,
        final GraphLayoutCache aGraphLayoutCache,
        final MorphingLayouter.MyListener listener,
        final boolean doExpand) {

        if (doExpand) {
            aGraph.getGraphLayoutCache().expand(
                new Object[]{cell});
            aGraph.getOverviewGraph().getGraphLayoutCache().expand(
                new Object[]{cell});
            aGraph.getOverviewGraph().getSelectionModel().clearSelection();
        }
        Rectangle2D newBounds =
//NEW           graphLayoutCache.getMapping(cell, false).getBounds();
            GraphConstants.getBounds(cell.getAttributes());
        Rectangle2D lastBounds = newBounds;

        if (MyGraphConstants.isCollapsed(cell.getAttributes())) {

            Map transportMap = new HashMap();
            AttributeMap newAttributes = new AttributeMap();
            transportMap.put(cell, newAttributes);

            // change collapsed attribute...
            if (doExpand) {
                MyGraphConstants.setCollapsed(newAttributes, false);
            }
            lastBounds =
                MyGraphConstants.getExpandedBounds(cell.getAttributes());
            if (lastBounds == null) {
                lastBounds = newBounds;
            } else {
                newBounds = new Rectangle2D.Double(
                    newBounds.getX(),
                    newBounds.getY(),
                    lastBounds.getWidth(),
                    lastBounds.getHeight());
            }

            if (doExpand) {
                GraphConstants.setBounds(newAttributes, newBounds);
                edit(transportMap, null, null, null);

                new MorphingLayouter(aGraph, listener, true)
                    .expandCompositeState(cell, newBounds.getBounds());
            } else {
                Map transportMap2 = new HashMap();
                AttributeMap newAttributes2 = new AttributeMap();
                transportMap2.put(cell, newAttributes2);

                MyGraphConstants.setExpandedBounds(newAttributes2,
                    new AttributeMap.SerializableRectangle2D(
                        newBounds.getX(), newBounds.getY(),
                        lastBounds.getWidth(), lastBounds.getHeight()));
                edit(transportMap2, null, null, null);
            }
        }

        final Point2D delta = new Point2D.Double(
            defaultDelta.getX() + newBounds.getX() - lastBounds.getX(),
            defaultDelta.getY() + newBounds.getY() - lastBounds.getY());

        // Now check the children...
        for (int i = 0; i < cell.getChildCount(); i++) {

            Map transportMap = new Hashtable();
            AttributeMap newAttributes = new AttributeMap();
            transportMap.put(cell.getChildAt(i), newAttributes);

            // contents have to be moved to the new pos
            newBounds = GraphConstants.getBounds(((DefaultGraphCell) cell
                    .getChildAt(i)).getAttributes()).getBounds2D();

            if (newBounds != null) { // == null for DefaultPorts
                newBounds = new AttributeMap.SerializableRectangle2D(
                    newBounds.getX() + delta.getX(),
                    newBounds.getY() + delta.getY(),
                    newBounds.getWidth(),
                    newBounds.getHeight());
                GraphConstants.setBounds(newAttributes, newBounds);
            }
            edit(transportMap, null, null, null);

            // now expand all children
            if (cell.getChildAt(i) instanceof CompositeStateCell) {
                CompositeStateCell child =
                    (CompositeStateCell) cell.getChildAt(i);
                if (!MyGraphConstants.isCollapsed(child.getAttributes())) {
                    expand(aGraph, child, delta, aGraphLayoutCache,
                        listener, doExpand);
                }
            } else if (cell.getChildAt(i) instanceof TransitionCell) {
                List points = GraphConstants.getPoints(
                    ((TransitionCell) cell.getChildAt(i)).getAttributes());
                List newPoints = new ArrayList();
                for (int pCount = 0; pCount < points.size(); pCount++) {

                    Point2D point = (Point2D) points.get(pCount);
                    newPoints.add(new AttributeMap.SerializablePoint2D(
                        point.getX() + delta.getX(),
                        point.getY() + delta.getY()));
                }
                transportMap = new Hashtable();
                newAttributes = new AttributeMap();
                transportMap.put(cell.getChildAt(i), newAttributes);
                GraphConstants.setPoints(newAttributes, newPoints);
                edit(transportMap, null, null, null);
```

```
            }
        }
    }

    /**
     * .
     * @param transitionCell .
     * @param aCurrentUserObject .
     * @throws CompoundLabelException .
    public void postEditTransitionLabel(final DefaultGraphCell
     transitionCell,
        final String aCurrentUserObject) throws CompoundLabelException {

        String currentUserObject = aCurrentUserObject;

        String labelString = (String) transitionCell.getUserObject();
        if (labelString == null) {
            labelString = "";
        }
        logFile.log(0, "labelString:" + labelString);
        // labelString has this format:
        // # trigger {condition} / effect
        // /
        // trigger
        // {condition}
        // trigger {condition}

        String triggerPart = null;
        String conditionPart = null;
        String effectPart = null;
        boolean flag = false;
        boolean slashFound = false;

        labelString = labelString.trim();

        // Cut off obsolete ending "/"...
        if (labelString.trim().endsWith("/")) {
            labelString = labelString.substring(
                labelString.length() - 1);
        }

        // Cut off starting "#"...
        flag = labelString.startsWith("#");
        if (flag) {
            labelString = labelString.substring(
                labelString.indexOf("#") + 1).trim();
        }

        slashFound = labelString.indexOf("/") != -1;
        if (slashFound) {
            effectPart = labelString.substring(
                labelString.indexOf("/") + 1).trim();
        }

        if (labelString.indexOf("{") != -1 && labelString.indexOf("}") !=
     -1) {
            conditionPart = labelString.substring(
                labelString.indexOf("{"), labelString.indexOf("}") + 1);
        }

        if (conditionPart != null) {
            triggerPart = labelString.substring(
                0, labelString.indexOf("{")).trim();
        } else if (slashFound) {
            triggerPart = labelString.substring(
                0, labelString.indexOf("/")).trim();
        } else {
            triggerPart = labelString;
        }

/*
        StringTokenizer st = new StringTokenizer(labelString);
        while (st.hasMoreTokens()) {
            String token = st.nextToken();
            if (token.equals("#")) {
                flag = true;
            } else if (token.startsWith("{") && token.endsWith("}")) {
                conditionPart = token;
            } else if (token.equals("/")) {
                slashFound = true;
            } else if (slashFound) {
                effectPart = token;
            } else {
                triggerPart = token;
            }
        }
      // Create the objects...
        setTransitionLabel(transitionCell, null, triggerPart, conditionPart,
            effectPart, flag, currentUserObject);
    }
*/

    /**
     * .
     * @param transitionCell .
     * @param triggerPart .
     * @param conditionPart .
     * @param effectPart .
     * @param isImmediate .
     * @param aCurrentUserObject .
     * @throws CompoundLabelException .
     */
    public void setTransitionLabel(final DefaultGraphCell transitionCell,
        final String triggerPart,
        final String conditionPart, final String effectPart,
        final boolean isImmediate, final String aCurrentUserObject)
        throws CompoundLabelException {
```

```
        String currentUserObject = aCurrentUserObject;

        Transition transition =
            (Transition) MyGraphConstants.getGraphicalObject(
                transitionCell.getAttributes());
        CompoundLabelParser.setStateChart(getCurrentStateChart());
        CompoundLabelParser.setCompositeState(
            (CompositeState) transition.getSource().getParent());
        CompoundLabel compoundLabel = new CompoundLabel();
        try {
            if (triggerPart != null && triggerPart.length() > 0) {
                logFile.log(2, "Trigger part: " + triggerPart);
                compoundLabel.setTrigger(
                    CompoundLabelParser.parseDelayExpression(
                        triggerPart, isImmediate));
            }
            if (conditionPart != null && conditionPart.length() > 0) {
                logFile.log(2, "Condition part: " + conditionPart);
                compoundLabel.setCondition(
                    CompoundLabelParser.parseBooleanExpression(
                        conditionPart));
            }
            if (effectPart != null && effectPart.length() > 0) {
                logFile.log(2, "Effect part: " + effectPart);
                compoundLabel.setEffect(
                    CompoundLabelParser.parseActions(effectPart));
            }

            // Use the old label layout...
            if (transition.getLabel() != null) {
                LabelLayoutInformation layout =
                    kielView.getLayoutInformation(transition.getLabel());
                if (layout != null) {
                    kielView.put(compoundLabel.getID(), layout);
                }
            }
            transition.setLabel(compoundLabel);
            updateKielTransitionLabelsPosition(
                (TransitionCell) transitionCell, null);

            currentUserObject = compoundLabel.toString();
            logFile.log(0, "set new CompoundLabel: "
                + compoundLabel.toString());
        } finally {
            AttributeMap newAttributes = new AttributeMap();
            GraphConstants.setValue(newAttributes, currentUserObject);
            Map transportMap = new HashMap();
            transportMap.put(transitionCell, newAttributes);
            edit(transportMap, null, null, null);
        }
    }

    /**
     * Update the position if no layout Info is available (the label is new
     !).
     * @param transitionCell .
     * @param newLabelPosition onCreate = newPosition == null. Coordinates
     are
     * Coordinates are not used!
     */
    public void updateKielTransitionLabelsPosition(
        final TransitionCell transitionCell,
        final Point2D newLabelPosition) {

        updateKielTransitionPriorityLabelPosition(
            transitionCell);
        updateKielTransitionLabelPosition(
            transitionCell, newLabelPosition);
    }

    /**
     * Update the position if no layout Info is available (the label is new
     !).
     * @param transitionCell .
     * @param newLabelPosition onCreate = newPosition == null. Coordinates
     are
     * Coordinates are not used!
     */
    private void updateKielTransitionLabelPosition(
        final TransitionCell transitionCell,
        final Point2D newLabelPosition) {

        final boolean onlyOnCreate = newLabelPosition == null;
        Transition transition = (Transition) MyGraphConstants
            .getGraphicalObject(transitionCell.getAttributes());

        if (transition.getLabel() == null
            || transition.getLabel().toString().length() == 0) {
            return;
        }
        LabelLayoutInformation layout =
            kielView.getLayoutInformation(transition.getLabel());
        if (layout == null) {
            layout = new LabelLayoutInformation();
            kielView.setLayoutInformation(transition.getLabel(), layout);
        } else if (onlyOnCreate) {
            return;
        }

        // calculate the label position...
        Point2D offset = MyGraphModelUtilities.calculateOffset(
            transition.getSource(), kielView);
        NodeCell sourceCell = (NodeCell) ((DefaultPort)
            ((TransitionCell) transitionCell).getSource())
            .getParent();
        NodeCell targetCell = (NodeCell) ((DefaultPort)
            ((TransitionCell) transitionCell).getTarget())
```

```java
            .getParent();
        if (sourceCell != targetCell) {
            Rectangle2D sourceBounds = GraphConstants.getBounds(
                sourceCell.getAttributes());
            Rectangle2D targetBounds = GraphConstants.getBounds(
                targetCell.getAttributes());
            layout.setCenterPoint(new Point(
                (int) (sourceBounds.getCenterX()
                + (targetBounds.getCenterX()
                    - sourceBounds.getCenterX()) / 2.0 - offset.getX()),
                (int) (sourceBounds.getCenterY()
                + (targetBounds.getCenterY()
                    - sourceBounds.getCenterY()) / 2.0 - offset.getY())));
        } else {
            List points = GraphConstants.getPoints(
                transitionCell.getAttributes());
            if (points.size() > 1) {
                layout.setCenterPoint(new Point(
                    (int) (((Point2D) points.get(1)).getX()
                        - offset.getX()),
                    (int) (((Point2D) points.get(1)).getY()
                        - offset.getY())));
            }
        }
        layout.setWidth(
            Properties.getWidth(Properties.getLabelFont(),
                transition.getLabel() + ""));
        layout.setHeight(Properties.getLabelFontSize());
    }

    /**
     * Update the position if no layout Info is available (the label is new
     !).
     * @param transitionCell .
     */
     private void updateKielTransitionPriorityLabelPosition(
        final TransitionCell transitionCell) {

        Transition transition = (Transition) MyGraphConstants
            .getGraphicalObject(transitionCell.getAttributes());

        LabelLayoutInformation layout =
            kielView.getLayoutInformation(transition.getPriority());
        if (layout == null) {
            layout = new LabelLayoutInformation();
            kielView.setLayoutInformation(transition.getPriority(), layout);
        }
        // calculate the label position...
        if (graphLayoutCache.getMapping(transitionCell, false) == null) {
            System.out.println("mapping is empty");
            return;
        }
        Rectangle2D start = graphLayoutCache.getMapping(transitionCell,
     false)
            .getBounds().getBounds2D().createIntersection(
                graphLayoutCache.getMapping(
                    getSourceVertex(this, transitionCell), false).getBounds
     ());
        if (start.isEmpty()) {
            System.out.println("is empty");
            return;
        }
        System.out.println("intersection:" + start);
        Point2D offset = MyGraphModelUtilities.calculateOffset(
            transition.getSource(), kielView);
        layout.setCenterPoint(new Point(
            (int) (start.getX() - offset.getX()),
            (int) (start.getY() - offset.getY())));
        layout.setWidth(
            Properties.getWidth(Properties.getLabelFont(),
                "<" + transition.getPriority().getValue() + ">"));
        layout.setHeight(Properties.getLabelFontSize());
    }

    /**
     * .
     * @param cell .
     * @param label .
     */
    public void setNodeLabel(final NodeCell cell, final String label) {
        AttributeMap newAttributes = new AttributeMap();
        GraphConstants.setValue(newAttributes, label);
        Map transportMap = new HashMap();
        transportMap.put(cell, newAttributes);
        edit(transportMap, null, null, null);
    }

    /**
     * .
     * @param cell .
     * @param oldBounds .
     */
    public void andStateCellEnlarged(final ANDStateCell cell,
        final Rectangle2D oldBounds) {

        Rectangle2D newBounds = GraphConstants.getBounds(cell.getAttributes
     ());

        if (oldBounds.getWidth() == newBounds.getWidth()
            && oldBounds.getHeight() == newBounds.getHeight()) {
            return;
        }

        Map transportMap = new HashMap();

        for (int i = 0; i < cell.getChildCount(); i++) {

            if (cell.getChildAt(i) instanceof RegionCell
```

```java
                || cell.getChildAt(i) instanceof DelimiterLineCell) {

                Rectangle2D childsBounds = GraphConstants.getBounds(
                    ((DefaultGraphCell) cell.getChildAt(i)).getAttributes())
;

                AttributeMap.SerializableRectangle2D changedBounds
                    = new AttributeMap.SerializableRectangle2D(
                        childsBounds.getX(), childsBounds.getY(),
                        childsBounds.getWidth(), childsBounds.getHeight());

                // left edge changed?
                if (oldBounds.getX() != newBounds.getX()) {
                    if (matches(oldBounds.getX(), childsBounds.getX())) {
                        changedBounds.setX(newBounds.getX());
                        changedBounds.setWidth(childsBounds.getWidth()
                            + newBounds.getWidth() - oldBounds.getWidth());
                    }
                // right edge changed?
                } else if (oldBounds.getWidth() != newBounds.getWidth()) {
                    if (matches(oldBounds.getX() + oldBounds.getWidth(),
                        childsBounds.getX() + childsBounds.getWidth())) {

                        changedBounds.setWidth(childsBounds.getWidth()
                            + newBounds.getWidth() - oldBounds.getWidth());
                    }
                }
                // upper edge changed?
                if (oldBounds.getY() != newBounds.getY()) {
                    if (matches(oldBounds.getY()
                        + Properties.getCompositeStateHeaderHeight(),
                        childsBounds.getY())) {

                        changedBounds.setY(childsBounds.getY()
                            + newBounds.getY() - oldBounds.getY());
                        changedBounds.setHeight(childsBounds.getHeight()
                            + newBounds.getHeight() - oldBounds.getHeight()
;
                    }
                // lower edge changed?
                } else if (oldBounds.getHeight() != newBounds.getHeight()) {
                    if (matches(oldBounds.getY() + oldBounds.getHeight(),
                        childsBounds.getY() + childsBounds.getHeight())) {

                        changedBounds.setHeight(childsBounds.getHeight()
                            + newBounds.getHeight() - oldBounds.getHeight())
;
                    }
                }
                AttributeMap newAttributes = new AttributeMap();
                GraphConstants.setBounds(newAttributes, changedBounds);
                transportMap.put(cell.getChildAt(i), newAttributes);
            }
        }
        edit(transportMap, null, null, null);
    }

    /**
     * .
     * @param cell .
     * @param oldBounds .
     * @param aGraphLayoutCache .
     */
    public void delimiterLineCellMoved(final DelimiterLineCell cell,
        final Rectangle2D oldBounds,
        final GraphLayoutCache aGraphLayoutCache) {

        Rectangle2D newBounds = GraphConstants.getBounds(cell.getAttributes
 ());
        Rectangle2D parentsBounds = GraphConstants.getBounds(
            ((DefaultGraphCell) cell.getParent()).getAttributes());
        final boolean isHorizontal =
            newBounds.getWidth() > newBounds.getHeight();
        Map transportMap = new HashMap();
        // Correct coordinates. Horizontal delimiter lines cannot change
        // their start.x coordinate! and they cannot leave their parent
        AttributeMap.SerializableRectangle2D changedBounds
            = new AttributeMap.SerializableRectangle2D(
                newBounds.getX(), newBounds.getY(),
                newBounds.getWidth(), newBounds.getHeight());
        boolean isOutside = false;

        if (isHorizontal) {
            changedBounds.setX(oldBounds.getX());

            if (newBounds.getY() < parentsBounds.getY()
                    + Properties.getCompositeStateHeaderHeight()
                || newBounds.getY() > parentsBounds.getY()
                    + parentsBounds.getHeight()) {

                changedBounds.setY(oldBounds.getY());
                isOutside = true;
            }
        } else {
            changedBounds.setY(oldBounds.getY());

            if (newBounds.getX() < parentsBounds.getX()
                || newBounds.getX() > parentsBounds.getX()
                    + parentsBounds.getWidth()) {

                changedBounds.setX(oldBounds.getX());
                isOutside = true;
            }
        }
        AttributeMap newAttributes = new AttributeMap();
        GraphConstants.setBounds(newAttributes, changedBounds);
        transportMap.put(cell, newAttributes);
        edit(transportMap, null, null, null);
```

```java
        if (isOutside) {
            return;
        }
        for (int i = 0; i < cell.getParent().getChildCount(); i++) {

            if (cell.getParent().getChildAt(i) instanceof DelimiterLineCell
                && cell.getParent().getChildAt(i) != cell) {

                Rectangle2D childsBounds = GraphConstants.getBounds(
                    ((DefaultGraphCell) cell.getParent()
                        .getChildAt(i)).getAttributes());
                changedBounds = new AttributeMap.SerializableRectangle2D(
                    childsBounds.getX(), childsBounds.getY(),
                    childsBounds.getWidth(), childsBounds.getHeight());

                boolean childIsHorizontal =
                    childsBounds.getWidth() > childsBounds.getHeight();

                if (isHorizontal && !childIsHorizontal) {
                    if (matches(oldBounds.getY(), childsBounds.getY())) {
                        changedBounds.setY(newBounds.getY());
                        changedBounds.setHeight(
                            changedBounds.getHeight()
                            + oldBounds.getY() - newBounds.getY());
                    } else if (matches(oldBounds.getY(),
                                childsBounds.getY()
                                + childsBounds.getHeight())) {
                        changedBounds.setHeight(
                            changedBounds.getHeight()
                            - oldBounds.getY() + newBounds.getY());
                    }
                } else if (!isHorizontal && childIsHorizontal) {
                    if (matches(oldBounds.getX(), childsBounds.getX())) {
                        changedBounds.setX(newBounds.getX());
                        changedBounds.setWidth(
                            changedBounds.getWidth()
                            + oldBounds.getX() - newBounds.getX());
                    } else if (matches(oldBounds.getX(),
                            childsBounds.getX() + childsBounds.getWidth()))
 {
                        changedBounds.setWidth(
                            changedBounds.getWidth()
                            - oldBounds.getX() + newBounds.getX());
                    }
                }
                newAttributes = new AttributeMap();
                GraphConstants.setBounds(newAttributes, changedBounds);
                transportMap.put(cell.getParent().getChildAt(i),
                    newAttributes);
            } else if (cell.getParent().getChildAt(i) instanceof RegionCell)
 {

                Rectangle2D childsBounds = GraphConstants.getBounds(
                    ((DefaultGraphCell) cell.getParent()
                        .getChildAt(i)).getAttributes());
                changedBounds = new AttributeMap.SerializableRectangle2D(
                    childsBounds.getX(), childsBounds.getY(),
                    childsBounds.getWidth(), childsBounds.getHeight());
                if (isHorizontal) { // horizontal delimiter...
                    // upper region?
                    if (matches(oldBounds.getY(),
                        childsBounds.getY()
                        + childsBounds.getHeight())) {

                        changedBounds.setHeight(childsBounds.getHeight()
                            + newBounds.getY() - oldBounds.getY());
                    // lower region?
                    } else if (matches(oldBounds.getY(), childsBounds.getY()
)) {

                        changedBounds.setY(newBounds.getY());
                        changedBounds.setHeight(childsBounds.getHeight()
                            + oldBounds.getY() - newBounds.getY());
                    }
                } else {    // vertical delimiter...
                    // left region?
                    if (matches(oldBounds.getX(),
                        childsBounds.getX() + childsBounds.getWidth())) {

                        changedBounds.setWidth(childsBounds.getWidth()
                            + newBounds.getX() - oldBounds.getX());
                    // right region?
                    } else if (matches(oldBounds.getX(), childsBounds.getX()
)) {

                        changedBounds.setX(newBounds.getX());
                        changedBounds.setWidth(childsBounds.getWidth()
                            + oldBounds.getX() - newBounds.getX());
                    }
                }
                newAttributes = new AttributeMap();
                GraphConstants.setBounds(newAttributes, changedBounds);
                transportMap.put(cell.getParent().getChildAt(i),
                    newAttributes);
            }
        }
        edit(transportMap, null, null, null);

        for (int i = 0; i < cell.getParent().getChildCount(); i++) {
            if (cell.getParent().getChildAt(i) instanceof RegionCell) {
                for (int childCount = 0;
                    childCount < cell.getParent().getChildAt(i)
                    .getChildCount(); childCount++) {

                    changeParent(
                        (DefaultGraphCell) cell.getParent()
                        .getChildAt(i).getChildAt(childCount),
                        aGraphLayoutCache);
```

kiel.editor.MyGraphModelUtilities

```java
package kiel.editor;

import java.awt.Dimension;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import kiel.dataStructure.ANDState;
import kiel.dataStructure.Choice;
import kiel.dataStructure.CompositeState;
import kiel.dataStructure.DeepHistory;
import kiel.dataStructure.DynamicChoice;
import kiel.dataStructure.FinalANDState;
import kiel.dataStructure.FinalORState;
import kiel.dataStructure.FinalSimpleState;
import kiel.dataStructure.FinalState;
import kiel.dataStructure.ForkConnector;
import kiel.dataStructure.History;
import kiel.dataStructure.InitialState;
import kiel.dataStructure.Join;
import kiel.dataStructure.Junction;
import kiel.dataStructure.Node;
import kiel.dataStructure.ORState;
import kiel.dataStructure.Region;
import kiel.dataStructure.SimpleState;
import kiel.dataStructure.State;
import kiel.dataStructure.Suspend;
import kiel.dataStructure.SynchState;
import kiel.dataStructure.Transition;
import kiel.editor.graph.ANDStateCell;
import kiel.editor.graph.DelimiterLineCell;
import kiel.editor.graph.GraphicalObjectCell;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.graph.NodeCell;
import kiel.editor.graph.ORStateCell;
import kiel.editor.graph.TransitionCell;
import kiel.editor.resources.Preferences;
import kiel.graphicalInformations.EdgeLayoutInformation;
import kiel.graphicalInformations.LinetoPath;
import kiel.graphicalInformations.NodeLayoutInformation;
import kiel.graphicalInformations.PathElement;
import kiel.graphicalInformations.Point;
import kiel.graphicalInformations.Properties;
import kiel.graphicalInformations.View;
import kiel.util.EdgeHelper;
import kiel.util.LinesDoNotIntersectException;
import kiel.util.LogFile;

import org.jgraph.graph.AttributeMap;
import org.jgraph.graph.CellView;
import org.jgraph.graph.DefaultGraphCell;
import org.jgraph.graph.GraphCell;
import org.jgraph.graph.GraphConstants;
import org.jgraph.graph.GraphLayoutCache;
import org.jgraph.graph.ParentMap;
import org.jgraph.graph.PortView;

/**
 * Contains static methods mainly used by MyGraphModel.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.20 $ last modified $Date: 2005/07/27 22:20:49 $
 */
public final class MyGraphModelUtilities {

    /**
     * The logging handle. Prints logs to standard out.
     */
    private static LogFile logFile = new LogFile("Editor", 2);

    /**
     *
     */
    private MyGraphModelUtilities() {
    }

    /**
     * .
     * @param kielPoint .
     * @param offset .
     * @param result .
     */
    static void addPoint(final Point kielPoint, final Point2D offset,
        final List result) {
        if (kielPoint != null) {
            Point2D jgraphPoint = new AttributeMap.SerializablePoint2D(
                kielPoint.getX() + offset.getX(),
                kielPoint.getY() + offset.getY());
            debug(true, "Add a point to jgraph: " + jgraphPoint, null);
            if (!jgraphPoint.equals(TransitionCell.DUMMY_POINT)
                && !result.contains(jgraphPoint)) {
                result.add(jgraphPoint);
            }
        }
    }

    /**
     * @param node .
     * @param aKielView .
```

```java
     * @return .
     */
    public static Point2D calculateOffset(final Node node,
        final View aKielView) {
        if (node.getParent() == null) {
            return new AttributeMap.SerializablePoint2D(
                Preferences.getInitialGraphOffset(),
                Preferences.getInitialGraphOffset());
            // Root ist etwas versetzt
        } else {
            Point2D parentOffset = calculateOffset(
                node.getParent(),
                aKielView);
            NodeLayoutInformation layout = aKielView.getLayoutInformation(
 node
                .getParent());
            return new AttributeMap.SerializablePoint2D(parentOffset.getX()
                + layout.getXPos(),
                parentOffset.getY() + layout.getYPos());
        }
    }

    /**
     * Transforms the KIEL transition points of the given transition to
     * JGraph transition points.
     * @param transition .
     * @param view .
     * @return .
     */
    public static ArrayList convertTransitionPointsKielToJGraph(
        final Transition transition, final View view) {

        debug(true, "convertTransitionPointsKielToJGraph..........", null);

        Point2D offset = calculateOffset(transition.getSource(), view);
        ArrayList result = new ArrayList();
        EdgeLayoutInformation layout =
            view.getLayoutInformation((Transition) transition);
        if (layout != null) {
            ArrayList points = layout.getAllPathElements();
            debug(true, "create JGraphPoints from these original kiel points
 :",
                null);
            debug(true, "", points);
            for (int i = 0; i < points.size(); i++) {
                addPoint(((PathElement) points.get(i)).getTargetPoint(),
                    offset, result);
//              addPoint(((PathElement) points.get(i)).getControlPoint1
    (),
//                  offset, result);
//              addPoint(((PathElement) points.get(i)).getControlPoint2
    (),
//                  offset, result);
            }
        }
        if (result.size() == 0) {
            result.add(TransitionCell.DUMMY_POINT);
            result.add(TransitionCell.DUMMY_POINT);
        }
        debug(true, "Created JGraph Points:", result);
        return result;
    }

    /**
     * .
     * @param cell .
     * @param kielView just for debug
     * @param transition just for debug
     * @return .
     */
    public static ArrayList convertTransitionPointsJGraphToKiel(
        final TransitionCell cell, final View kielView,
        final Transition transition) {

        debug(true, "convertTransitionPointsJGraphToKiel..........", null);
        // 1. Create KIEL-Points.......
        Rectangle2D parentBounds = GraphConstants.getBounds(
            ((DefaultGraphCell) cell.getParent()).getAttributes());
        debug(true, "Parent:" + cell.getParent() + ":" + parentBounds, null)
 ;
        ArrayList jgraphPoints =
            (ArrayList) GraphConstants.getPoints(cell.getAttributes());
        boolean debug = true;
        debug(debug, "JGRAPH Points:", jgraphPoints);

        ArrayList kielPoints = new ArrayList();
        for (int i = 1; i < jgraphPoints.size() - 1; i++) {
            // first and last are DUMMY_POINTs or the end points

            if (!TransitionCell.DUMMY_POINT.equals(jgraphPoints.get(i))) {

                kielPoints.add(new kiel.graphicalInformations.Point(
                        (int) (((Point2D) jgraphPoints.get(i)).getX()
                            - parentBounds.getX()),
                        (int) (((Point2D) jgraphPoints.get(i)).getY()
                            - parentBounds.getY())));
            }
        }
        debug(debug, "Old KIEL points:", kielView.getLayoutInformation(
            transition).getAllPathElements());

        // 2. Create Path-Elements......
        debug(debug, "New KIEL points:", kielPoints);

        try {
//          if (GraphConstants.getLineStyle(cell.getAttributes())
//              == GraphConstants.STYLE_ORTHOGONAL) {
//
```

```java
//                kielView.setLayoutInformation(transition,
//                    EdgeHelper.convertPolyPath(kielPoints, transition
//                        .getSource(), transition.getTarget(), kielView));
//            } else {
                kielView.setLayoutInformation(transition,
                    EdgeHelper.convertSplinePath(kielPoints, transition
                        .getSource(), transition.getTarget(), kielView));
//            }
        } catch (LinesDoNotIntersectException e) {
            e.printStackTrace();
        }
        debug(debug, "New points after convertPolyPath:", kielView
            .getLayoutInformation(transition).getAllPathElements());

        // 3. Correct Path-Elements......
        ArrayList allKielPoints = kielView.getLayoutInformation(
            transition).getAllPathElements();

        if (allKielPoints.size() > 2 && allKielPoints.get(
            allKielPoints.size() - 1) instanceof LinetoPath) {
            allKielPoints.remove(allKielPoints.size() - 1);
        }
        if (allKielPoints.size() > 2 && allKielPoints.get(1)
            instanceof LinetoPath) {
            allKielPoints.remove(1);
        }
        debug(debug, "New points corrected:", kielView.getLayoutInformation(
            transition).getAllPathElements());
        return kielPoints;
    }

    /**
     * .
     * @param debug .
     * @param msg .
     * @param list .
     */
    private static void debug(final boolean debug, final String msg,
        final ArrayList list) {
        if (!debug) {
            return;
        }
//        System.out.println(msg);
//        for (int i = 0; list != null && i < list.size(); i++) {
//            System.out.println(list.get(i));
//        }
  }

    /**
     * Creates KIEL datastructure objects, for example: given ANDState.class
       as
     * nodeClass will return an ANDState object. This could easily done
     * with java reflection as well, but this techique is generally avoided
       in
     * the editor code.
     * @param nodeClass .
     * @return .
     */
    static Node createKIELNodeFor(final Class nodeClass) {

        Node node = null;
        if (nodeClass == ANDState.class) {
            node = new ANDState();
        } else if (nodeClass == Choice.class) {
            node = new Choice();
        } else if (nodeClass == DeepHistory.class) {
            node = new DeepHistory();
        } else if (nodeClass == DynamicChoice.class) {
            node = new DynamicChoice();
        } else if (nodeClass == FinalANDState.class) {
            node = new FinalANDState();
        } else if (nodeClass == FinalORState.class) {
            node = new FinalORState();
        } else if (nodeClass == FinalSimpleState.class) {
            node = new FinalSimpleState();
        } else if (nodeClass == FinalState.class) {
            node = new FinalState();
        } else if (nodeClass == ForkConnector.class) {
            node = new ForkConnector();
        } else if (nodeClass == History.class) {
            node = new History();
        } else if (nodeClass == InitialState.class) {
            node = new InitialState();
        } else if (nodeClass == Join.class) {
            node = new Join();
        } else if (nodeClass == Junction.class) {
            node = new Junction();
        } else if (nodeClass == ORState.class) {
            node = new ORState();
        } else if (nodeClass == Region.class) {
            node = new Region();
        } else if (nodeClass == SimpleState.class) {
            node = new SimpleState();
        } else if (nodeClass == State.class) {
            node = new State();
        } else if (nodeClass == Suspend.class) {
            node = new Suspend();
        } else if (nodeClass == SynchState.class) {
            node = new SynchState();
        } else {
            logFile.log(2, "Unexpected kiel type:" + nodeClass
                + " instantiate dynamically...");
            try {
                node = (Node) nodeClass.newInstance();
            } catch (InstantiationException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
```

```
            }
        }
        return node;
    }

    /**
     * Helper method.
     * @param cell .
     * @param useDefaultBoundsForCollapsedStates .
     * @return .
     */
    static Rectangle2D getBoundsOf(final Object cell,
        final boolean useDefaultBoundsForCollapsedStates) {

        Rectangle2D bounds = GraphConstants.getBounds(
                ((DefaultGraphCell) cell).getAttributes());
        if (useDefaultBoundsForCollapsedStates
            && MyGraphConstants.isCollapsed(
                ((DefaultGraphCell) cell).getAttributes())) {
            bounds = bounds.getBounds2D();
            bounds.setRect(
                bounds.getX(),
                bounds.getY(),
                Properties.getDefaultCompositeStateSize().getWidth(),
                Properties.getDefaultCompositeStateSize().getHeight());
        }
        return bounds;
    }

    /**
     * Returns the highest priority number of the given node's outgoing
     * transitions.
     * @param node .
     * @return .
     */
    static int getHighestPriority(final Node node) {

        Transition[] transitions = (Transition[]) node.
     getOutgoingTransitions()
                .toArray(new Transition[0]);
        int maxPriority = 0;
        for (int i = 0; i < transitions.length; i++) {
            if (transitions[i].getPriority().getValue() > maxPriority) {
                maxPriority = transitions[i].getPriority().getValue();
            }
        }
        return maxPriority;
    }

    /**
     * The initial dimension is used for the node creation. It is stored in
     the
     * user Preferences.
     * TODO What is with subclasses?
     * @param elementClass .
     * @return .
     */
    public static Dimension getInitialDimensionOf(final Class elementClass)
     {

        if (elementClass == SimpleState.class) {
            return new Dimension(
                Preferences.getDefaultState().width,
                Preferences.getDefaultState().height);
        } else if (elementClass == FinalSimpleState.class) {
            return new Dimension(
                Preferences.getDefaultNode().width,
                Preferences.getDefaultNode().height);
        } else if (elementClass == InitialState.class) {
            return new Dimension(
                Preferences.getDefaultNode().width,
                Preferences.getDefaultNode().height);
        } else if (elementClass == Suspend.class) {
            return new Dimension(
                Preferences.getDefaultNode().width,
                Preferences.getDefaultNode().height);
        } else if (elementClass == DynamicChoice.class) {
            return new Dimension(
                Preferences.getDefaultNode().width,
                Preferences.getDefaultNode().height);
        } else if (elementClass == History.class) {
            return new Dimension(
                Preferences.getDefaultNode().width,
                Preferences.getDefaultNode().height);
        } else if (elementClass == DeepHistory.class) {
            return new Dimension(
                Preferences.getDefaultNode().width,
                Preferences.getDefaultNode().height);
        } else if (elementClass == ORState.class) {
            return new Dimension(
                Properties.getDefaultCompositeStateSize().width,
                Properties.getDefaultCompositeStateSize().height);
        } else if (elementClass == FinalState.class) {
            return new Dimension(
                Preferences.getDefaultState().width,
                Preferences.getDefaultState().height);
        } else {
            return new Dimension(
                Preferences.getDefaultNode().width,
                Preferences.getDefaultNode().height);
        }
    }

    /**
     * JGraph stores absolute coordinates. This method returns the relative
     * coordinates, relative to its parent. This method just subracts the
     given
     * cell's coordinates from its parent's coordinates.
```

```
     * @param cell .
     * @return .
     */
    static kiel.graphicalInformations.Point getRelativeCoordinates(
        final GraphicalObjectCell cell) {

        Rectangle2D bounds = GraphConstants.getBounds(
            cell.getAttributes()).getBounds();

        NodeCell parent;
        if (cell instanceof NodeCell) {
            parent = (NodeCell) ((NodeCell) cell).getParent();
        } else {
            parent = (NodeCell) ((DelimiterLineCell) cell).getParent();
        }
        Rectangle2D parentBounds = null;
        if (parent == null) {
            parentBounds = new AttributeMap.SerializableRectangle2D(
                Preferences.getInitialGraphOffset(),
                Preferences.getInitialGraphOffset(), 0, 0);
        } else {
            parentBounds = GraphConstants.getBounds(
                parent.getAttributes()).getBounds();
        }
        return new kiel.graphicalInformations.Point(
                (int) (bounds.getX() - parentBounds.getX()),
                (int) (bounds.getY() - parentBounds.getY()));
    }
    /**
     * Returns the smallest node of the given viewClass, containing the
     given
     * bounds. Never returns a PortView!
     * @param viewClass .
     * @param bounds .
     * @param graphLayoutCache .
     * @param checkForContains if true then use Rectangle.contains() else
     use
     * Rectangle.intersect().
     * @param excludedCell .
     * @return .
     */
    public static CellView getSmallestViewOfType(final Class viewClass,
        final Rectangle2D bounds, final GraphLayoutCache graphLayoutCache,
        final boolean checkForContains,
        final DefaultGraphCell excludedCell) {

        CellView[] views = graphLayoutCache.getCellViews();
        CellView newSmallestParent = null;
        CellView excludedView = null;
        if (excludedCell != null) {
            excludedView = graphLayoutCache.getMapping(excludedCell, false);
        }
        for (int i = 0; i < views.length; i++) {
            if (views[i] instanceof PortView || views[i] == excludedView) {
                continue;
            }
            if (viewClass.isInstance(views[i])) {
                if (
                   (checkForContains && views[i].getBounds().contains(bounds
 ))
                   || (!checkForContains
                        && views[i].getBounds().intersects(bounds))) {

                    if (newSmallestParent == null) {
                        newSmallestParent = views[i];
                    } else if (newSmallestParent.getBounds().getWidth()
                            * newSmallestParent.getBounds().getHeight()
                        > views[i].getBounds().getWidth()
                            * views[i].getBounds().getHeight()) {
                        newSmallestParent = views[i];
                    }
                }
            }
        }
        return newSmallestParent;
    }

    /**
     * Returns the valid area of a composite state, means: where can we
     place a
     * child? There are two calculations, one for ANDStates and ORStates (
     don't
     * place within the header) and Regions.
     * TODO could we replace Preferences.getValidAreaBorder() with
     * Preferences.getIntersectionSensitivity()?
     * @param cell .
     * @return .
     */
    public static Rectangle2D getValidArea(final DefaultGraphCell cell) {

        Rectangle2D validArea = getBoundsOf(cell, false).getBounds2D();
        validArea.setRect(
            validArea.getX() + Preferences.getValidAreaBorder(),
            validArea.getY() + Preferences.getValidAreaBorder(),
            validArea.getWidth() - 2 * Preferences.getValidAreaBorder(),
            validArea.getHeight() - 2 * Preferences.getValidAreaBorder());

        if (cell instanceof ORStateCell || cell instanceof ANDStateCell) {
            validArea.setRect(
                validArea.getX(),
                validArea.getY()
//              + Properties.getCompositeStateHeaderHeight(),
                + Properties.getUpperOffset((State) MyGraphConstants
                    .getGraphicalObject(cell.getAttributes())),
                validArea.getWidth(),
                validArea.getHeight()
//              - Properties.getCompositeStateHeaderHeight());
                - Properties.getUpperOffset((State) MyGraphConstants
```

```java
                    .getGraphicalObject(cell.getAttributes()));
        }
        return validArea;
    }

    /**
     * According to the information stored in the parent map the KIEL
     * datastructure is updated.
     * @param pm .
     */
    static void updateKielParent(final ParentMap pm) {
        if (pm == null) {
            return;
        }
        Iterator entries = pm.entries();
        while (entries.hasNext()) {

            ParentMap.Entry entry = (ParentMap.Entry) entries.next();
            if (entry.getChild() instanceof NodeCell
                && entry.getParent() != null) {
                // can be null to remove a parent
                Node child = (Node) MyGraphConstants.getGraphicalObject(
                    ((GraphCell) entry.getChild()).getAttributes());
                Node currentParent = child.getParent();
                Node newParent = (Node) MyGraphConstants.getGraphicalObject(
                    ((GraphCell) entry.getParent()).getAttributes());
                // remove from old Parent...
                if (currentParent instanceof CompositeState) {
                    ((CompositeState) currentParent).removeSubnode(child);
                }
                // add to new Parent...
                if (newParent instanceof CompositeState
                    && !((CompositeState) newParent).getSubnodes()
                        .contains(child)) {
                    ((CompositeState) newParent).addSubnode(child);
                }
            }
        }
    }

    /**
     * Checks if the MyGraphModel and the FileInterface did its job well :-)
     * TODO this method should be deleted. Everything already works fine!
     * @param node .
     */
    static void updateKielParentChildRelation(final Node node) {
        if (node instanceof CompositeState
                && ((CompositeState) node).getSubnodes() != null) {

            Node[] children = (Node[]) ((CompositeState) node).getSubnodes()
                .toArray(new Node[0]);
            for (int i = 0; children != null && i < children.length; i++) {
                if (children[i].getParent() != node) {
                    children[i].setParent((CompositeState) node);
                    logFile.log(2, "Editor corrected Parent Child relation
 .");
                }
            }
        }
    }
}
```

kiel.editor.MyJGraph

```java
package kiel.editor;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.Stroke;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.awt.print.PageFormat;
import java.awt.print.Printable;
import java.util.ArrayList;
import java.util.Arrays;

import javax.swing.JCheckBoxMenuItem;
import javax.swing.JComponent;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JPopupMenu;

import kiel.dataStructure.StateChart;
import kiel.editor.controller.ActionAddLoopTransition;
import kiel.editor.controller.ActionEditCollapsed;
import kiel.editor.controller.ActionEditConvertToBezier;
import kiel.editor.controller.ActionEditConvertFlipFinalAttribute;
import kiel.editor.controller.ActionEditConvertToNormalTransition;
import kiel.editor.controller.ActionEditConvertToORState;
import kiel.editor.controller.ActionEditConvertToOrthogonal;
import kiel.editor.controller.ActionEditConvertToSimpleState;
import kiel.editor.controller.ActionEditConvertToSpline;
import kiel.editor.controller.ActionEditConvertToStrongAbortion;
import kiel.editor.controller.ActionEditConvertToWeakAbortion;
import kiel.editor.controller.ActionEditCut;
import kiel.editor.controller.ActionEditEdit;
import kiel.editor.controller.ActionEditPaste;
import kiel.editor.controller.ActionEditRemove;
import kiel.editor.controller.ActionEditStateDoActions;
import kiel.editor.controller.ActionEditStateEntryActions;
import kiel.editor.controller.ActionEditStateEvents;
import kiel.editor.controller.ActionEditStateExitActions;
import kiel.editor.controller.ActionEditStateVariables;
import kiel.editor.controller.ActionEditTransitionPriority;
import kiel.editor.controller.EditorAction;
import kiel.editor.controller.EditorActions;
import kiel.editor.controller.MyMarqueeHandler;
import kiel.editor.graph.ANDStateCell;
import kiel.editor.graph.CompositeStateCell;
import kiel.editor.graph.ConditionalTransitionCell;
import kiel.editor.graph.DelimiterLineCell;
import kiel.editor.graph.DelimiterLineView;
import kiel.editor.graph.InitialArcCell;
import kiel.editor.graph.InitialStateCell;
import kiel.editor.graph.NodeCell;
import kiel.editor.graph.NormalTerminationCell;
import kiel.editor.graph.ORStateCell;
import kiel.editor.graph.RegionCell;
import kiel.editor.graph.SimpleStateCell;
import kiel.editor.graph.StateCell;
import kiel.editor.graph.StrongAbortionCell;
import kiel.editor.graph.SuspensionCell;
import kiel.editor.graph.TransitionCell;
import kiel.editor.graph.WeakAbortionCell;
import kiel.editor.resources.Preferences;
import kiel.editor.resources.ResourceLoader;
import kiel.graphicalInformations.Properties;
import kiel.graphicalInformations.View;

import org.jgraph.JGraph;
import org.jgraph.event.GraphModelEvent;
import org.jgraph.event.GraphModelListener;
import org.jgraph.event.GraphSelectionEvent;
import org.jgraph.event.GraphSelectionListener;
import org.jgraph.graph.CellView;
import org.jgraph.graph.DefaultEdge;
import org.jgraph.graph.DefaultGraphCell;
import org.jgraph.graph.DefaultGraphModel;
import org.jgraph.graph.DefaultPort;
import org.jgraph.graph.GraphCell;
import org.jgraph.graph.GraphConstants;
import org.jgraph.graph.GraphLayoutCache;
import org.jgraph.graph.GraphModel;
import org.jgraph.graph.PortRenderer;
import org.jgraph.graph.PortView;


/**
 * MyJGraph represents the graph drawing area of the editor component.
 * MyJGraph customizes the JGraph framework for the use of statecharts.
     Mainly
 * MyJGraph introduces different KIEL-specific diagram elements. Next,
     MyJGraph
 * differs from JGraph in that it contains a overview graph, displayed as
 * different layer, and that it contains an undo manager. Many properties of
 * MyJGraph can be adjusted through the Preferences class.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.88 $ last modified $Date: 2005/08/01 17:41:30 $
 */
public final class MyJGraph extends JGraph
    implements Preferences.Listener, Printable {
```

```
/**
 * This instance variable is only filled for the "main"-MyJGraph object.
 * A MyJGraph object, which contains an overviewGraph, is called a "main
 "-
 * MyJGraph object. The overview graph is not a "main"-MyJGraph object,
 * consequently it does not have an overviewGraph object itself. (=No
 * recursion!). The overview graph has the same content as its
 * "main"-MyJGraph, but it is scaled in a way that it always fits the
 * window size!
 */
private final MyJGraph overviewGraph;

/**
 * This reference is filled for overview graphs since a overview graph
 * needs to know which cell is currently selected.
 */
private MyJGraph mainGraph;

/**
 * @see #overviewGraph
 * @return .
 */
public MyJGraph getOverviewGraph() {
    return overviewGraph;
}

/**
 * The plus-image showed in the upper left corner of an Or-state and
 * And-State, indicating that this state is collapsed. Used for the main
 * graph.
 */
private static final Image PLUS_IMAGE =
    ResourceLoader.get("plus.gif").getImage();

/**
 * The minus-image showed in the upper left corner of an Or-state and
 * And-State indicating that this state is expanded. Used for the main
 * graph.
 */
private static final Image MINUS_IMAGE =
    ResourceLoader.get("minus.gif").getImage();

/**
 * The plus-image showed in the upper left corner of an Or-state and
 * And-State, indicating that this state is collapsed. Used for the
 * overview graph.
 */
private static final Image PLUS_IMAGE_GRAY =
    ResourceLoader.get("plusgray.gif").getImage();

/**
 * The minus-image showed in the upper left corner of an Or-state and
 * And-State indicating that this state is expanded. Used for the
 overview
 * graph.
 */
private static final Image MINUS_IMAGE_GRAY =
    ResourceLoader.get("minusgray.gif").getImage();

/**
 * Returns the either the black plus image (for the main graph) or the
 * grayscaled plus image for the overview graph.
 * @return .
 */
public Image getPlusImage() {
    if (overviewGraph == null) {
        return PLUS_IMAGE_GRAY;
    } else {
        return PLUS_IMAGE;
    }
}

/**
 * Returns the either the black minus image (for the main graph) or the
 * grayscaled minus image for the overview graph.
 * @return .
 */
public Image getMinusImage() {
    if (overviewGraph == null) {
        return MINUS_IMAGE_GRAY;
    } else {
        return MINUS_IMAGE;
    }
}

/**
 * Returns the main color for drawing states. it returns gray (depending
  on
 * the Preferences) for the overview
 * graph, returns red for an incomplete main graph element and returns
 * black for a complete element in the main graph.
 * @param view .
 * @return .
 */
public Color getViewsBlack(final CellView view) {
    if (overviewGraph == null) {
        return Preferences.getOverviewColor();
    } else {
        if (!(view.getCell() instanceof NodeCell)
            || isComplete((DefaultGraphCell) view.getCell())) {
            return Color.BLACK;
        } else {
            return getViewsRed();
        }
    }
}
```

```java
/**
 *
 * @param cell .
 * @return .
 */
private static boolean isComplete(final DefaultGraphCell cell) {
    if (cell instanceof ORStateCell) {
        return contains(cell, InitialStateCell.class) == 1
            && (cell.getParent() == null /* is root state */
                || hasIncomingTransitions(cell));
    } else if (cell instanceof ANDStateCell) {
        for (int i = 0; i < cell.getChildCount(); i++) {
            if (!isComplete((DefaultGraphCell) cell.getChildAt(i))) {
                return false;
            }
        }
        return hasIncomingTransitions(cell);
    } else if (cell instanceof RegionCell) {
        return contains(cell, InitialStateCell.class) == 1;
    } else if (cell instanceof SimpleStateCell) {
        return hasIncomingTransitions(cell);
    } else if (cell instanceof InitialStateCell) {
        return hasOutgoingTransitions(cell);
    } else {
        return true;
    }
}

/**
 *
 * @param cell .
 * @param childClass .
 * @return .
 */
private static int contains(final GraphCell cell,
    final Class childClass) {

    int count = 0;
    if (!(cell instanceof CompositeStateCell)) {
        return 0;
    } else {
        Object[] children =
            ((CompositeStateCell) cell).getChildren().toArray();
        for (int i = 0; i < children.length; i++) {
            if (childClass.isInstance(children[i])) {
                count++;
            }
        }
        return count;
    }
}

/**
 *
 * @param cell .
 * @return .
 */
private static boolean hasIncomingTransitions(final DefaultGraphCell
 cell) {
    if (cell.getParent() == null) {
        return false;
    }
    for (int i = 0; i < cell.getParent().getChildCount(); i++) {
        if (cell.getParent().getChildAt(i) instanceof TransitionCell) {
            TransitionCell transition =
                (TransitionCell) cell.getParent().getChildAt(i);
            if (((DefaultPort) transition.getTarget()).getParent()
                == cell) {
                return true;
            }
        }
    }
    return false;
}

/**
 *
 * @param cell .
 * @return .
 */
private static boolean hasOutgoingTransitions(final DefaultGraphCell
 cell) {
    if (cell.getParent() == null) {
        return false;
    }
    for (int i = 0; i < cell.getParent().getChildCount(); i++) {
        if (cell.getParent().getChildAt(i) instanceof TransitionCell) {
            TransitionCell transition =
                (TransitionCell) cell.getParent().getChildAt(i);
            if (((DefaultPort) transition.getSource()).getParent()
                == cell) {
                return true;
            }
        }
    }
    return false;
}

/**
 * returns dark green for a main graph element and gray (depending on
 the
 * Preferences) for an overview graph element.
 * @return .
 */
public Color getViewsGreen() {
    if (overviewGraph == null) {
        return Preferences.getOverviewColor();
```

```java
        } else {
            return Color.GREEN.darker();
        }
    }

    /**
     * returns red for a main graph element and gray (depending on the
     * Preferences) for an overview graph element.
     * @return .
     */
    public Color getViewsRed() {
        if (overviewGraph == null) {
            return Preferences.getOverviewColor();
        } else {
            return Color.RED;
        }
    }

    /**
     * returns gray for a main graph element and gray (depending on the
     * Preferences) for an overview graph element.
     * @return .
     */
    public Color getViewsGray() {
        if (overviewGraph == null) {
            return Preferences.getOverviewColor();
        } else {
            return Color.GRAY;
        }
    }

    /**
     * returns yellow for a main graph element and gray (depending on the
     * Preferences) for an overview graph element.
     * @return .
     */
    public Color getViewsYellow() {
        if (overviewGraph == null) {
            return Preferences.getOverviewColor();
        } else {
            return Color.YELLOW;
        }
    }

    /**
     *
     */
    private static final float[] DASH_PHASE = new float[] {5f, 5f};

    /**
     *
     */
    private static final float MITER_LIMIT = 10;

    /**
     *
     */
    private static final BasicStroke STROKE_HIGHLIGHT_DELIMITER =
        new BasicStroke(Properties.getLineWidth() + 1,
            BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER, MITER_LIMIT,
            DASH_PHASE, 0.0f);

    /**
     *
     */
    private static final BasicStroke STROKE_HIGHLIGHT_NODE =
        new BasicStroke(Properties.getLineWidth() + 1);

    /**
     *
     */
    private static final BasicStroke STROKE_DELIMITER =
        new BasicStroke(Properties.getLineWidth(),
            BasicStroke.CAP_BUTT, BasicStroke.JOIN_MITER, MITER_LIMIT,
            DASH_PHASE, 0.0f);

    /**
     *
     */
    private static final BasicStroke STROKE_NODE =
        new BasicStroke(Properties.getLineWidth());

    /**
     * Returns the main color for drawing states. it returns gray (depending
     on
     * the Preferences) for the overview
     * graph, returns red for an incomplete main graph element and returns
     * black for a complete element in the main graph.
     * @param view .
     * @return .
     */
    public Stroke getStroke(final CellView view) {
        if ((overviewGraph == null
            && Arrays.asList(mainGraph.getSelectionCells())
                .contains(view.getCell()))
         || (overviewGraph != null
             && Arrays.asList(getSelectionCells())
                 .contains(view.getCell()))) {

            if (view instanceof DelimiterLineView) {
                return STROKE_HIGHLIGHT_DELIMITER;
            } else {
                return STROKE_HIGHLIGHT_NODE;
            }
        } else {
            if (view instanceof DelimiterLineView) {
                return STROKE_DELIMITER;
            } else {
```

```
                return STROKE_NODE;
            }
        }
    }

    /**
     * Stores the Undo/Redo history and can perform undo and redo steps.
     */
    private final UndoClusterManager undoManager = new UndoClusterManager();

    /**
     * @see #undoManager
     * @return .
     */
    public UndoClusterManager getUndoClusterManager() {
        return undoManager;
    }

    /**
     * Updates all Preferences dependent properties of the graph drawing
     area.
     * @see kiel.editor.resources.Preferences.Listener#preferencesChanged()
     */
    public void preferencesChanged() {
        if (overviewGraph != null) {
            setGridEnabled(Preferences.getGridEnabled());
            setGridVisible(Preferences.getGridVisible());
            // TODO Should be a property...
            final int gridColor = 170;
            setGridColor(new Color(gridColor, gridColor, gridColor));
            setGridSize(Preferences.getGridSize());
            setTolerance(Preferences.getCellSelectionTolerance());
            setHandleColor(Preferences.getGuiColorDark());
            setHandleSize(Preferences.getSizeHandleSize());
            setHandleColor(Preferences.getSizeHandleColor());
        }
    }

    /**
     * Returns an instance of <code>MyJGraph</code> which displays the
     * specified data model.
     * @param model .
     * @param isMainGraph .
     */
    MyJGraph(final GraphModel model, final boolean isMainGraph) {

        super(model);
        Preferences.addListener(this);

        setOpaque(false);
        setPortsVisible(true); // Make Ports Visible by Default
        setInvokesStopCellEditing(true); // Accept edits if click on
     background
        setGraphLayoutCache(new GraphLayoutCache(
            getModel(), new MyDefaultCellViewFactory(), true));
        getGraphLayoutCache().setSelectsLocalInsertedCells(false);
        if (isMainGraph) {
            setMarqueeHandler(new MyMarqueeHandler(this));
            overviewGraph = new MyJGraph(getModel(), false);
            overviewGraph.mainGraph = this;
            overviewGraph.setPortsVisible(false);
            // Paint the current selected cell in the overview!
            getSelectionModel().addGraphSelectionListener(
                new GraphSelectionListener() {
                    public void valueChanged(final GraphSelectionEvent e) {
                        if (e.isAddedCell()) {
                            overviewGraph.repaint();
                        }
                    }
                }
            );
        } else {
            overviewGraph = null;
        }
        PortView.renderer = new PortRenderer() {
            {
                setForeground(Preferences.getPortColor());
            }
        };
        preferencesChanged();
    }

    /**
     * This is the hook for our own GraphUI class.
     * @see javax.swing.JComponent#updateUI()
     */
    public void updateUI() {
        setUI(new MyBasicGraphUI());
//        ((MyBasicGraphUI) getUI()).addEditListener(this);
    }

    /**
     * The Graph's popup menu.
     */
    private JPopupMenu popup = new JPopupMenu();

    /**
     * @see #popup
     * @return .
     */
    public boolean isPopupVisible() {
        return popup.isVisible();
    }
    /**
     * Updates the popup menu and its content, depending on which kind
     * of diagram element is selected.
     * @param x x-coordinate.
     * @param y y-coordinate.
```

```java
     */
    public void showPopup(final int x, final int y) {

        popup.removeAll();
        ArrayList allItems = getMenuItemsForSelected();
        for (int i = 0; i < allItems.size(); i++) {
            if (allItems.get(i) instanceof JComponent) {
                popup.add((JComponent) allItems.get(i));
            } else {
                popup.add((EditorAction) allItems.get(i));
            }
        }
        popup.show(this, x, y);
    }

    /**
     *
     */
    private JMenu layouterMenu;

    /**
     *.
     * @param aLayouterMenu .
     */
    public void setLayouterMenu(final JMenu aLayouterMenu) {
        layouterMenu = aLayouterMenu;
    }

    /**
     * Returns all popup or menu items for the selected element.
     * @return .
     */
    public ArrayList getMenuItemsForSelected() {

        ArrayList items = new ArrayList();

        final GraphCell selected = (GraphCell) getSelectionCell();

        if (selected != null) {
            if (!(selected instanceof DelimiterLineCell)
               /* && !(selected instanceof RegionCell)*/) {
                items.add(Utils.createJMenuItem(ActionEditEdit.class));
            }
            if (selected instanceof StateCell) {
                items.add(Utils.createJMenuItem(
                    ActionEditStateEntryActions.class));
                items.add(Utils.createJMenuItem(
                    ActionEditStateDoActions.class));
                items.add(Utils.createJMenuItem(
                    ActionEditStateExitActions.class));
                if (selected instanceof CompositeStateCell) {
                    items.add(Utils.createJMenuItem(
                        ActionEditStateEvents.class));
                    items.add(Utils.createJMenuItem(
                        ActionEditStateVariables.class));
                }
            }
//           items.add(EditorActions.get(ActionFileProperties.class));
            if (!(selected instanceof RegionCell)) {
                items.add(new JPopupMenu.Separator());
                items.add(Utils.createJMenuItem(ActionEditRemove.class));
            }
            items.add(new JPopupMenu.Separator());
            if (EditorActions.get(ActionEditCut.class).isEnabled()
                && !(selected instanceof TransitionCell)
                && !(selected instanceof DelimiterLineCell)
                && !(selected instanceof RegionCell)) {

                items.add(Utils.createJMenuItem(ActionEditCut.class));
            }
        }
        if (EditorActions.get(ActionEditPaste.class).isEnabled()
            && ((ActionEditCut) EditorActions.get(ActionEditCut.class))
                .getCurrentCutCell() != null) {
            items.add(Utils.createJMenuItem(ActionEditPaste.class));
        }
        if (selected != null) {
            if (selected instanceof ANDStateCell
                || selected instanceof ORStateCell) {
                items.add(layouterMenu);
            }
            if (getMyGraphModel().getRootCell() != selected) {

                if (selected instanceof CompositeStateCell
                        && !(selected instanceof RegionCell)) {
                    items.add(new JPopupMenu.Separator());
                    items.add(getMenuItemCollapsed());
                    items.add(new JPopupMenu.Separator());
                }
                if (selected instanceof SimpleStateCell) {
                    items.add(Utils.createJMenuItem(
                        ActionEditConvertToORState.class));
                }
                if (selected instanceof NodeCell
                    && !hasOutgoingTransitions(
                        (DefaultGraphCell) selected)) {
                    items.add(Utils.createJMenuItem(
                        ActionEditConvertFlipFinalAttribute.class));
                }
                if (selected.getClass() == ORStateCell.class) {
                    if (((ORStateCell) selected).getChildCount() == 1) {
                        items.add(Utils.createJMenuItem(
                            ActionEditConvertToSimpleState.class));
                    }
                }
                if (selected instanceof NodeCell) {
                    items.add(Utils.createJMenuItem(
                        ActionAddLoopTransition.class));
```

```
            }
        }
        if (selected instanceof DefaultEdge) {
            items.add(Utils.createJMenuItem(
                ActionEditTransitionPriority.class));
            if (GraphConstants.getLineStyle(
                selected.getAttributes()) != GraphConstants.STYLE_SPLINE
    ) {
                items.add(Utils.createJMenuItem(
                    ActionEditConvertToSpline.class));
            }
            if (GraphConstants.getLineStyle(
                selected.getAttributes()) != GraphConstants.STYLE_BEZIER
    ) {
                items.add(Utils.createJMenuItem(
                    ActionEditConvertToBezier.class));
            }
            if (GraphConstants.getLineStyle(selected.getAttributes())
                != GraphConstants.STYLE_ORTHOGONAL) {

                items.add(Utils.createJMenuItem(
                    ActionEditConvertToOrthogonal.class));
            }
            items.add(new JPopupMenu.Separator());
            if (!(selected instanceof ConditionalTransitionCell)
                && !(selected instanceof InitialArcCell)
                && !(selected instanceof SuspensionCell)) {

                if (!(selected instanceof NormalTerminationCell)) {
                    items.add(Utils.createJMenuItem(
                            ActionEditConvertToNormalTransition.class));
                }
                if (!(selected instanceof WeakAbortionCell)) {
                    items.add(Utils.createJMenuItem(
                            ActionEditConvertToWeakAbortion.class));
                }
                if (!(selected instanceof StrongAbortionCell)) {
                    items.add(Utils.createJMenuItem(
                            ActionEditConvertToStrongAbortion.class));
                }
            }
        }
    }
    return items;
}

 /**
 * This menu item always reflects the collapsed state of the currently
 * selected state.
 */
private JMenuItem menuItemCollapsed =
    new JCheckBoxMenuItem(EditorActions.get(ActionEditCollapsed.class));

/**
 * @return the menu item reflecting the collapsed state of the currently
 * selected state
 */
public JMenuItem getMenuItemCollapsed() {
    return menuItemCollapsed;
}

/**
 * Returns the coordinates of the last popup trigger, used for the popup
 * menu and for placing a new state.
 */
private Point2D currentPopupPoint;

/**
 * @see #currentPopupPoint
 * @param aCurrentPopupPoint The currentPopupPoint to set.
 */
public void setCurrentPopupPoint(final Point2D aCurrentPopupPoint) {
    this.currentPopupPoint = aCurrentPopupPoint;
}

/**
 * @see #currentPopupPoint
 * @return .
 */
public Point2D getCurrentPopupPoint() {
    return currentPopupPoint;
}

/**
 * Casts the current Model to MyGraphModel. If the current model is not
 a
 * MyGraphModel then null is returned
 * @return .
 */
public synchronized MyGraphModel getMyGraphModel() {
    if (getModel() instanceof MyGraphModel) {
        return ((MyGraphModel) getModel());
    } else {
        return null;
    }
}

/**
 * @see kiel.util.main.KielComponent#getCurrentStateChart()
 */
public StateChart getCurrentStateChart() {
    if (getMyGraphModel() != null) {
        return getMyGraphModel().getCurrentStateChart();
    } else {
        return null;
    }
}
```

```java
/**
 * @see kiel.util.main.KielComponent#getCurrentView()
 */
public View getCurrentView() {
    if (getMyGraphModel() != null) {
        return getMyGraphModel().getCurrentView();
    } else {
        return null;
    }
}

/**
 * After setting the model the cells have set visible, but that action
 * should not belog to the new edit cluster.
 */
private boolean isSettingModelCompleted;

/**
 * Don't call directly but always through editor.setModel().
 * @see org.jgraph.JGraph#setModel(org.jgraph.graph.GraphModel)
 */
public void setModel(final GraphModel model) {
    isSettingModelCompleted = false;
    super.setModel(model);

    if (overviewGraph != null) {

        overviewGraph.setModel(model);

        if (getUndoClusterManager() != null && getUndoClusterManager()
            .getUndoManager() != null) {
            getUndoClusterManager().clear();
            model.addUndoableEditListener(getUndoClusterManager()
                .getUndoManager());
        }
        updateOverviewGraphScale();
    }

    model.addGraphModelListener(new GraphModelListener() {

        public void graphChanged(final GraphModelEvent e) {

            if (!getUndoClusterManager().isUndoManagerInProgress()) {
                getGraphLayoutCache().setVisible(e.getChange()
                    .getInserted(), true);
                if (!isSettingModelCompleted) {
                    getUndoClusterManager().discardCurrentCluster();
                    isSettingModelCompleted = true;
                }
                updateOverviewGraphScale();
            }
        }
    });
    if (getGraphLayoutCache() != null) {
        getGraphLayoutCache().setVisible(
            DefaultGraphModel.getDescendants(getModel(), getRoots())
                .toArray(), true);
    }
}

/**
 * The overview graph should always fit the window size. So this method
 is
 * called whenever the graph has changed or when a new model is set or
 when
 * the window has been resized.
 */
public void updateOverviewGraphScale() {
    if (overviewGraph != null && getMyGraphModel() != null) {
        Rectangle2D bounds = GraphConstants.getBounds(
                getMyGraphModel().getRootCell().getAttributes());
        final int roundingTip = 20;
        double scaleFactorWidth =
            (overviewGraph.getWidth() - roundingTip) / bounds.getWidth()
;
        double scaleFactorHeight =
            (overviewGraph.getHeight() - roundingTip) / bounds.getHeight
();
        if (Math.min(scaleFactorWidth, scaleFactorHeight) < 1) {
            overviewGraph.setScale(
                Math.min(scaleFactorWidth, scaleFactorHeight));
        } else {
            overviewGraph.setScale(1);
        }
    }
}

/**
 * @see java.awt.print.Printable#print(java.awt.Graphics,
 * java.awt.print.PageFormat, int)
 */
public int print(final Graphics g, final PageFormat pageFormat,
    final int page) {

    double oldScale = getScale();
    try {
      // turn off double buffering of graph so we don't get bitmap
printed
      setDoubleBuffered(false);

      double pageWidth = pageFormat.getImageableWidth();
      double pageHeight = pageFormat.getImageableHeight();
      int cols = (int) (this.getWidth() / pageWidth) + 1;
      int rows = (int) (this.getHeight() / pageHeight) + 1;
      int pageCount = cols * rows;
      double xScale = pageWidth / this.getWidth();
      double yScale = pageHeight / this.getHeight();
```

```
          double scale = Math.min(xScale, yScale);

          double tx = 0.0;
          double ty = 0.0;
          final double half = 0.5;
          if (xScale > scale) {
            tx = half * (xScale - scale) * this.getWidth();
          } else {
            ty = half * (yScale - scale) * this.getHeight();
          }
          ((Graphics2D) g).translate((int) pageFormat.getImageableX(),
              (int) pageFormat.getImageableY());
          ((Graphics2D) g).translate(tx, ty);
          ((Graphics2D) g).scale(scale, scale);

          if (page >= pageCount) {
            return NO_SUCH_PAGE;
          }

          this.paint(g);
        } finally {
          //  turn double buffering back on
          setDoubleBuffered(true);
          setScale(oldScale);
        }
        return PAGE_EXISTS;
    }
}
```

kiel.editor.PreferencesDialog

```java
package kiel.editor;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;

import kiel.editor.resources.Preferences;
import kiel.editor.resources.ResourceBundle;

/**
 * Displays the user preferences. If the user has never saved his
     preferences
 * to disk, then the default preferences will be shown, but in any case the
 * current preferences will be shown, that means, the user can change
 * preferences in this dialog and these changes are applied immediately. but
 * if the user does not select save, then these changes will be lost when
 * exiting the application. An object of this class holds a reference to the
 * loaded preferences and a reference to the default preferences. when the
 * user changes a value, then the correctness of that value will be checked
 * the next time this value is applied. for most of the values this is will
 * be done immediately. if the check failed (for example, the user typed in
 * 3,4 instead of 3.4) then the wrong value is overwritten with the default
 * preferences value.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.22 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class PreferencesDialog extends JDialog
    implements Preferences.Listener {

    /**
     * @param owner the parent frame for the dialog
     *
     */
    public PreferencesDialog(final JFrame owner) {
        super(owner, "", true);

        Preferences.addListener(this);

        Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
        final int dialogSize = 400;
        final int headerHight = 32;
        setBounds(
            (d.width - dialogSize) / 2,
            (d.height - dialogSize) / 2,
            dialogSize,
            dialogSize);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        getContentPane().setLayout(new BorderLayout());

        JPanel headerPanel = Utils.createJPanel(new BorderLayout(), false);
        headerPanel.setPreferredSize(new Dimension(0, headerHight));
        headerLabel.setOpaque(false);
        headerPanel.add(headerLabel);

        getContentPane().add(headerPanel, "North");

        preferencesChanged();
    }

    /**
     * .
     */
    private JLabel headerLabel = new JLabel();

    /**
     * whenever the user presses the "save" button, the data will be saved
     to
     * disk only if the user changed a value beforehand.
     */
    private boolean somethingChanged;

    /**
     * the current table is stored for removing it to reflect immediately.
     * the change the preferences language
     */
    private JScrollPane currentTable;

    /**
     * @see #currentTable
     */
    private JPanel currentButtonPanel;

    /**
     * Creates a table with dynamically read translated Preferences keys and
     * values.
     * @return .
     */
    private JScrollPane createTable() {
```

```java
        final String[] keys =
            (String[]) Preferences.keySet().toArray(new String[0]);
        Object[][] data = new Object[keys.length][2];
        for (int i = 0; i < data.length; i++) {
            data[i] = new String[] {
                ResourceBundle.getString(keys[i]),
                Preferences.getProperty(keys[i])};
        }
        return currentTable = new JScrollPane(
            new JTable(data, new String[]{
                ResourceBundle.getString("preferencesKey"),
                ResourceBundle.getString("preferencesValue")}) {

                {
                    setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
                    final int column0 = 244;
                    final int column1 = 130;
                    getColumnModel().getColumn(0).setPreferredWidth(column0)
  ;
                    getColumnModel().getColumn(1).setPreferredWidth(column1)
  ;
                }
                    public void setValueAt(final Object value, final int row
  ,
                        final int col) {

                        if (!Preferences.getProperty(keys[row]).equals(value
  )) {
                            somethingChanged = true;
                        }
                        Preferences.setProperty(keys[row], (String) value);
                        super.setValueAt(value, row, col);
                    }
                }
            );
    }

    /**
     * Whenever the user changes the language, the whole table and button
     panel
     * have to created again.
     * @see kiel.editor.resources.Preferences.Listener#preferencesChanged()
     */
    public void preferencesChanged() {
        if (currentTable != null) {
            getContentPane().remove(currentTable);
            getContentPane().remove(currentButtonPanel);
        }
        setTitle(ResourceBundle.getString("PreferencesDialog"));
        headerLabel.setText(getTitle());
        getContentPane().add(createTable(), "Center");
        getContentPane().add(createButtonPanel(), "South");
        validate();
        repaint();
    }

    /**
     * creates the button panel with cancel and save button.
     * @return .
     */
    private JPanel createButtonPanel() {
        final int gap = 6;
        currentButtonPanel =
            Utils.createJPanel(new FlowLayout(FlowLayout.RIGHT, gap, gap));

        JButton resetButton =
            new JButton(ResourceBundle.getString("defaultValues"));
        resetButton.addActionListener(new ActionListener() {
            public void actionPerformed(final ActionEvent e) {
                Preferences.setDefaultValues();
                dispose();
            }
        });
        resetButton.setOpaque(false);
        currentButtonPanel.add(resetButton);

        JButton okButton =
            new JButton(ResourceBundle.getString("ActionFileSave"));
        okButton.addActionListener(new ActionListener() {
            public void actionPerformed(final ActionEvent e) {
                if (somethingChanged) {
                    Preferences.store();
                }
                dispose();
            }
        });
        okButton.setOpaque(false);
        currentButtonPanel.add(okButton);

        JButton cancelButton = new JButton(ResourceBundle.getString("Cancel
  "));
        cancelButton.addActionListener(new ActionListener() {
            public void actionPerformed(final ActionEvent e) {
                dispose();
            }
        });
        cancelButton.setOpaque(false);
        currentButtonPanel.add(cancelButton);
        return currentButtonPanel;
    }
}
```

kiel.editor.TraceLogDialog

```java
package kiel.editor;

import java.awt.BorderLayout;
import java.awt.Component;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintWriter;
import java.text.SimpleDateFormat;
import java.util.Date;

import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;
import javax.swing.JTable;
import javax.swing.table.DefaultTableCellRenderer;

import kiel.editor.resources.Preferences;
import kiel.editor.resources.ResourceBundle;
import kiel.editor.resources.ResourceLoader;

/**
 * This dialog displays all undo/redo steps as well as the whole edit step
 * history.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.21 $ last modified $Date: 2005/07/25 11:44:37 $
 */
public final class TraceLogDialog extends JDialog {

    /**
     * The given current undoable edit steps.
     */
    private EditStep[] undoableEditSteps;

    /**
     * The given current redoable edit steps.
     */
    private EditStep[] redoableEditSteps;

    /**
     * The given all edit steps (complet history).
     */
    private EditStep[] allEditSteps;

    /**
     * .
     */
    private Editor editor;

    /**
     * Creates a dialog with size 400x400 with a tabbed pane inside.
     * @param anEditor .
     * @param anAllUndoableEditSteps .
     * @param anAllRedoableEditSteps .
     * @param anAllEditSteps .
     */
    public TraceLogDialog(final Editor anEditor,
        final EditStep[] anAllUndoableEditSteps,
        final EditStep[] anAllRedoableEditSteps,
        final EditStep[] anAllEditSteps) {

        super(anEditor.getKielFrame().getJFrame(), "Trace Log Dialog", true)
;
        undoableEditSteps = anAllUndoableEditSteps;
        redoableEditSteps = anAllRedoableEditSteps;
        allEditSteps = anAllEditSteps;
        editor = anEditor;

        final int dialogSize = 400;
        final int headerHight = 32;
        Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
        setBounds(
            (d.width - dialogSize) / 2,
            (d.height - dialogSize) / 2,
            dialogSize,
            dialogSize);
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);

        getContentPane().setLayout(new BorderLayout());

        JPanel headerPanel = Utils.createJPanel(new BorderLayout(), false);
        headerPanel.setPreferredSize(new Dimension(0, headerHight));
        JLabel label = new JLabel("Trace Log Dialog");
        label.setOpaque(false);
        headerPanel.add(label);

        getContentPane().add(headerPanel, "North");
        getContentPane().add(createContent(), "Center");
        getContentPane().add(createButtonPanel(), "South");
    }
```

```java
    /**
     * Formats the time of the iven edit step in HH:mm:ss.
     * @param editStep .
     * @return .
     */
    private String formatTime(final EditStep editStep) {
        return new SimpleDateFormat("HH:mm:ss").format(
            new Date(editStep.getCurrentTimeMillis()));
    }

    /**
     * Reads the class name associated to the edit step, cuts it, and
     returns
     * the translation of the cutted name via the ResourceBundle.
     * @param editStep .
     * @return .
     */
    private String formatAction(final EditStep editStep) {
        return ResourceBundle.getString(
            editStep.getEditorAction().getClass().getName().substring(
            editStep.getEditorAction().getClass().getName().lastIndexOf('.')
            + 1));
    }

    /**
     *
     */
    private JTabbedPane tabPane = new JTabbedPane();

    /**
     * Creates the Tabbed pane with its content.
     * @return .
     */
    private JTabbedPane createContent() {

        Object[][] data =
            new Object[undoableEditSteps.length + redoableEditSteps.length
     ][];
        for (int i = 0; i < undoableEditSteps.length; i++) {
            data[i] = new String[] {
                formatTime(undoableEditSteps[i]),
                formatAction(undoableEditSteps[i])};
        }
        for (int i = 0; i < redoableEditSteps.length; i++) {
            data[undoableEditSteps.length + i] = new String[] {
                formatTime(redoableEditSteps[redoableEditSteps.length - i -
     1]),
                formatAction(
                    redoableEditSteps[redoableEditSteps.length - i - 1])};
        }
        JTable table = new JTable(data, new String[]{
            ResourceBundle.getString("time"),
            ResourceBundle.getString("editStep")});
        table.setDefaultRenderer(Object.class, new DefaultTableCellRenderer
 () {
            public Component getTableCellRendererComponent(final JTable
 table,
                final Object value, final boolean isSelected,
                final boolean hasFocus, final int row, final int column) {

                Component c = super.getTableCellRendererComponent(table,
 value,
                    isSelected, hasFocus, row, column);
                if (row < undoableEditSteps.length) {
                    c.setBackground(Preferences.getGuiColorLight());
                } else {
                    c.setBackground(Preferences.getGuiColorDark());
                }
                return c;
            }

        });
        JScrollPane currentTable = new JScrollPane(table);
        tabPane.insertTab(
            ResourceBundle.getString("undoSteps"),
            ResourceLoader.get("Undo16.gif"),
            currentTable,
            ResourceBundle.getString("tooltipUndoSteps"), 0);

        data = new Object[allEditSteps.length][];
        for (int i = 0; i < data.length; i++) {
            data[i] = new String[] {
                formatTime(allEditSteps[i]),
                formatAction(allEditSteps[i]),
                ResourceBundle.getString(allEditSteps[i].getType())};
        }
        currentTable = new JScrollPane(
            new JTable(data, new String[]{
                ResourceBundle.getString("time"),
                ResourceBundle.getString("editStep"),
                ResourceBundle.getString("editStepType")}));
        tabPane.insertTab(
            ResourceBundle.getString("allEditSteps"),
            ResourceLoader.get("Edit16.gif"),
            currentTable,
            ResourceBundle.getString("tooltipAllEditSteps"), 1);

        return tabPane;
    }

    /**
     * Creates the button panel with the export and cancel button.
     * @return .
     */
    private JPanel createButtonPanel() {
        final int gap = 6;
        JPanel currentButtonPanel =
```

```
            Utils.createJPanel(new FlowLayout(FlowLayout.RIGHT, gap, gap));
        JButton okButton =
            new JButton(ResourceBundle.getString("ActionFileExport"));
        okButton.addActionListener(new ActionListener() {
            public void actionPerformed(final ActionEvent e) {
                JFileChooser chooser = new JFileChooser(
                    ResourceBundle.getString("TraceLogSaveDialog"));
                setCancelButtonText(chooser,
                    ResourceBundle.getString("Cancel"));
                int answer = chooser.showDialog(
                    TraceLogDialog.this.getOwner(),
                    ResourceBundle.getString("ActionFileSave"));
                if (answer == JFileChooser.APPROVE_OPTION) {
                    try {
                        PrintWriter out = new PrintWriter(new
FileOutputStream(
                            chooser.getSelectedFile()), true);
                        if (tabPane.getSelectedIndex() == 0) {
                            for (int i = 0; i < undoableEditSteps.length; i
++) {
                                out.println(
                                    formatTime(undoableEditSteps[i])
                                    + Preferences.getTraceLogSeparator()
                                    + formatAction(undoableEditSteps[i]));
                            }
                            for (int i = 0; i < redoableEditSteps.length; i
++) {
                                out.println(
                                    formatTime(redoableEditSteps[i])
                                    + Preferences.getTraceLogSeparator()
                                    + formatAction(redoableEditSteps[i]));
                            }
                        } else {
                            for (int i = 0; i < allEditSteps.length; i++) {
                                out.println(
                                    formatTime(allEditSteps[i])
                                    + Preferences.getTraceLogSeparator()
                                    + formatAction(allEditSteps[i])
                                    + Preferences.getTraceLogSeparator()
                                    + ResourceBundle.getString(
                                        allEditSteps[i].getType()));
                            }
                        }
                        out.close();
                    } catch (FileNotFoundException ex) {
                        editor.getKielFrame().handleException(
                            ResourceBundle.getString("ErrorOnSaving"),
                            false, ex);
                    }
                }
                dispose();
            }
        });
        okButton.setOpaque(false);
        currentButtonPanel.add(okButton);
        JButton cancelButton = new JButton(ResourceBundle.getString("Cancel
"));
        cancelButton.addActionListener(new ActionListener() {
            public void actionPerformed(final ActionEvent e) {
                dispose();
            }
        });
        cancelButton.setOpaque(false);
        currentButtonPanel.add(cancelButton);
        return currentButtonPanel;
    }

    /**
     * Workaround for setting the cancel button text of a component that
     * has a "Abbrechen" or "Cancel" button.
     * @param c e.g. a JFileChooser object
     * @param text the text for the cancel button
     */
    private static void setCancelButtonText(final Container c,
        final String text) {

        for (int i = 0; i < c.getComponents().length; i++) {
            if (c.getComponents()[i] instanceof JButton) {
                if ("Cancel".equals(((JButton) c.getComponents()[i]).getText
 ())
                    || "Abbrechen".equals(
                        ((JButton) c.getComponents()[i]).getText())) {
                        ((JButton) c.getComponents()[i]).setText(text);
                }
            } else if (c.getComponents()[i] instanceof Container) {
                setCancelButtonText((Container) c.getComponents()[i], text);
            }
        }
    }
}
```

kiel.editor.UndoClusterManager

```java
package kiel.editor;

import java.util.Stack;

import javax.swing.JSlider;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;
import javax.swing.event.UndoableEditEvent;
import javax.swing.undo.UndoManager;

import kiel.editor.controller.ActionEditRedo;
import kiel.editor.controller.ActionEditUndo;
import kiel.editor.controller.EditorAction;
import kiel.editor.controller.EditorActions;
import kiel.util.LogFile;

import org.jgraph.graph.GraphUndoManager;

/**
 * The UndoClusterManager manages the the EditorActions, the user edit steps
    .
 * The GraphUndoManager manages the GraphModel change steps. The
 * UndoClusterManager wraps the GraphUndoManager to cluster and wrap the
 * GraphModel change steps. That means, the GraphModel change steps during
    an
 * EditorAction step have to be counted and this count, wrapped into an
 * EditStep object, has to be stored on a stack. When the user wants to
 * perform an undo, he calls UndoClusterManager.undo(), which takes an
    EditStep
 * containing a count n from the stack, and calls n times
 * GraphUndoManager.undo(). The same with redo steps.
 * The UndoClusterManager also provides a Slider object, unfortunately this
 * slider currently just acts as visible component without working features
    :-(
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.43 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class UndoClusterManager {

    /**
     * The GraphUndoManager object for handling undo operations as
     GraphModel
     * change steps.
     */
    private UndoManager undoManager;

    /**
     * .
     */
    private UndoClusterManagerListener listener;

    /**
     * .
     * @param aListener .
     */
    void setListener(final UndoClusterManagerListener aListener) {
        listener = aListener;
    }

    /**
     * The sliderChangeListener reacts on using the slider knob to perform
     undo
     * or redo steps.
     */
    private ChangeListener sliderChangeListener = new ChangeListener() {
        public void stateChanged(final ChangeEvent e) {
            if (!updateSlider) {
                return;
            }
            if (undoableEditHappenedClusters.size() > getSlider().getValue()
 ) {
                while (undoableEditHappenedClusters.size()
                    > getSlider().getValue()) {
                    undo();
                }
            } else {
                while (undoableEditHappenedClusters.size()
                        + redoableEditHappenedClusters.size()
                    > getSlider().getValue()) {
                    redo();
                }
            }
        }
    };

    /**
     * The sliderChangeListener should not react, when the slider is
     adjusted
     * inside the UndoClusterManager.
     */
    private boolean updateSlider = true;

    /**
     * Initializes the slider and performs a clear() call.
     *
     */
    UndoClusterManager() {
        getSlider().setMinimum(0);
        getSlider().setMaximum(0);
        getSlider().addChangeListener(sliderChangeListener);
```

```java
        clear();
    }

    /**
     *
     * @return the UndoManager responsible for GraphModel change steps
     */
    UndoManager getUndoManager() {
        return undoManager;
    }

    /**
     * The slider displaying the current relation between undo and redo
     steps.
     */
    private final JSlider slider = Utils.createJSlider();

    /**
     * @see #slider
     * @return .
     */
    JSlider getSlider() {
        return slider;
    }

    /**
     *
     * @return the content of the undo EditStep stack
     */
    public EditStep[] getAllUndoableEditStepsAscending() {
        return (EditStep[]) undoableEditHappenedClusters.toArray(
            new EditStep[0]);
    }

    /**
     *
     * @return the content of the redo EditStep stack
     */
    public EditStep[] getAllRedoableEditStepsAscending() {
        return (EditStep[]) redoableEditHappenedClusters.toArray(
            new EditStep[0]);
    }

    /**
     * Update Undo/Redo Button State and slider dependent on the undo and
     redo
     * stack.
     */
    private void updateHistoryButtonsAndSlider() {
        EditorActions.get(ActionEditUndo.class).setEnabled(
            !undoableEditHappenedClusters.isEmpty());
        EditorActions.get(ActionEditRedo.class).setEnabled(
            !redoableEditHappenedClusters.isEmpty());
        updateSlider = false;
        getSlider().setMinimum(0);
        getSlider().setMaximum(undoableEditHappenedClusters.size()
            + redoableEditHappenedClusters.size());
        getSlider().setValue(undoableEditHappenedClusters.size());
        updateSlider = true;

        if (listener != null) {
            listener.stackChanged(
                undoableEditHappenedClusters.size(),
                redoableEditHappenedClusters.size());
        }
    }

    /**
     * the count of GraphModel change steps of the current EditStep.
     */
    private int undoableEditHappenedCount;

    /**
     * the undo EditStep stack.
     */
    private final Stack undoableEditHappenedClusters = new Stack();

    /**
     * clears the whole GraphModel undo history by destroying the old
     * GraphUndoManager. With this also the EditStep undo and redo stacks
     have
     * to be cleared.
     *
     */
    void clear() {
        redoableEditHappenedClusters.clear();
        undoableEditHappenedClusters.clear();
        undoableEditHappenedCount = 0;
        undoManager = new GraphUndoManager() {

            public void undoableEditHappened(final UndoableEditEvent e) {
                if (isUndoManagerInProgress()) {
                    return;
                }

                super.undoableEditHappened(e);
                updateHistoryButtonsAndSlider(); // Update Toolbar
                undoableEditHappenedCount++;
            }
        };
        undoManager.setLimit(Integer.MAX_VALUE);
        updateHistoryButtonsAndSlider();
    }

    /**
     * Process the whole undo stack.
     *
```

```
 */
public void undoAll() {
    while (!undoableEditHappenedClusters.isEmpty()) {
        undo();
    }
}

/**
 * Redo the whole undo stack.
 *
 */
public void redoAll() {
    while (!redoableEditHappenedClusters.isEmpty()) {
        redo();
    }
}

/**
 * Perform one EditorAction EditStep redo.
 */
public void redo() {
    if (redoableEditHappenedClusters.isEmpty()) {
        return;
    }
    isUndoManagerInProgress = true;
    int count = ((EditStep) undoableEditHappenedClusters
        .push(redoableEditHappenedClusters.pop())).getCount();
    for (int i = 0; i < count; i++) {
        // Redo the last Change to the Model or the View
        undoManager.redo();
    }
    isUndoManagerInProgress = false;
    updateHistoryButtonsAndSlider();
}

/**
 * Perform an undo and do not put this EditStep on the redo stack, so
 this
 * step will be lost = non redoable.
 */
public void nonRedoableUndo() {
    undo();
    if (!redoableEditHappenedClusters.isEmpty()) {
        redoableEditHappenedClusters.pop();
    }
}

/**
 * The logging handle. Prints logs to standard out.
 */
private LogFile logFile = new LogFile("Editor", 2);

/**
 * Perform one EditorAction EditStep undo.
 */
public void undo() {
    if (undoableEditHappenedClusters.isEmpty()) {
        return;
    }
    isUndoManagerInProgress = true;
    int count = ((EditStep) redoableEditHappenedClusters
        .push(undoableEditHappenedClusters.pop())).getCount();
    for (int i = 0; i < count; i++) {
        // Undo the last Change to the Model or the View
        try {
            undoManager.undo();
        } catch (NullPointerException e) {
            logFile.log(1,
                "UndoClusterManager.undo() Already reported JGraph-bug")
;
        }
    }
    isUndoManagerInProgress = false;
    updateHistoryButtonsAndSlider();
}

/**
 * is true whenever the UndoClusterManager.undo() or redo() method is
 * processed.
 */
private boolean isUndoManagerInProgress;

/**
 * @return @see inProgress
 */
boolean isUndoManagerInProgress() {
    return isUndoManagerInProgress;
}

/**
 * Tells the UndoClusterManager to start counting the GraphModel change
 * steps because an EditorAction has just begun.
 */
public void setUndoClusterStart() {
    undoableEditHappenedCount = 0;
}

/**
 * Throw away the information how many GraphModel change steps have been
 * done in the current EditStep. This method should only be called if
 the
 * undo steps on the GraphModel are done by someone else, for example,
 when
 * the JGraph framework does them internally.
 */
public void discardCurrentCluster() {
    setUndoClusterStart();
}
```

```
    /**
     * the redo EditStep stack.
     */
    private final Stack redoableEditHappenedClusters = new Stack();

    /**
     * Tells the UndoClusterManager that the current EditorAction (EditStep)
     * has ended. So, a new EditStep object will be created and put on the
     undo
     * stack.
     * @param editorAction .
     */
    public void setUndoClusterEnd(final EditorAction editorAction) {
        if (undoableEditHappenedCount > 0) {
            undoableEditHappenedClusters.push(new EditStep(
                undoableEditHappenedCount, editorAction,
                EditStep.TYPE_SUCCESSFUL));
            undoableEditHappenedCount = 0;
            redoableEditHappenedClusters.clear();
            updateHistoryButtonsAndSlider();
        }
    }

    /**
     *
     * @return true if the redo stack is not empty
     */
    public boolean canRedo() {
        return !redoableEditHappenedClusters.isEmpty();
    }

    /**
     *
     * @return true if the undo stack is not empty
     */
    public boolean canUndo() {
        return !undoableEditHappenedClusters.isEmpty();
    }
}
```

kiel.editor.Utils

```
package kiel.editor;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Container;
import java.awt.GradientPaint;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.LayoutManager;
import java.awt.Paint;
import java.awt.event.ContainerEvent;
import java.awt.event.ContainerListener;
import java.awt.event.KeyEvent;
import java.awt.event.MouseEvent;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;

import javax.swing.AbstractButton;
import javax.swing.Action;
import javax.swing.JButton;
import javax.swing.JComboBox;
import javax.swing.JComponent;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JPanel;
import javax.swing.JScrollBar;
import javax.swing.JScrollPane;
import javax.swing.JSlider;
import javax.swing.JSplitPane;
import javax.swing.JTabbedPane;
import javax.swing.JToggleButton;
import javax.swing.JToolBar;
import javax.swing.KeyStroke;
import javax.swing.event.MouseInputListener;
import javax.swing.plaf.ButtonUI;
import javax.swing.plaf.metal.MetalButtonUI;
import javax.swing.plaf.metal.MetalToggleButtonUI;
import javax.swing.plaf.metal.MetalToolBarUI;

import kiel.editor.controller.EditorActions;
import kiel.editor.resources.Preferences;
import kiel.editor.resources.ResourceBundle;
import kiel.editor.resources.ResourceLoader;

/**
 * Defines a special look&feel for some swing classes, providing two color
 * gradient background paints, and provides a Splitpane and TabbedPane with
 * floatable (draggable) child components.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.42 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class Utils {

    /**
     * Not for instantiation.
     *
     */
    private Utils() { }

    /**
     *
     * @param c .
     * @param g .
     * @param topToBottom .
     */
    private static void paintComponent(final JComponent c, final Graphics g,
        final boolean topToBottom) {

        paintComponent(
            c,
            g,
            topToBottom,
            Preferences.getGuiColorLight(),
            Preferences.getGuiColorDark());
    }

    /**
     *
     * @param c .
     * @param g .
     * @param topToBottom .
     * @param light .
     * @param dark .
     */
    private static void paintComponent(final JComponent c, final Graphics g,
        final boolean topToBottom, final Color light, final Color dark) {

        Paint oldPaint = ((Graphics2D) g).getPaint();

        if (topToBottom) {
            ((Graphics2D) g).setPaint(new GradientPaint(0, 0,
                light, 0, c.getSize().height, dark, true));
        } else {
            ((Graphics2D) g).setPaint(new GradientPaint(0, 0,
                dark, c.getSize().width, 0, light, true));
        }
```

```java
        ((Graphics2D) g).fillRect(0, 0, c.getSize().width, c.getSize().
    height);
        ((Graphics2D) g).setPaint(oldPaint);
    }

    /**
     * .
     * @param title .
     * @param shortcut .
     * @param icon .
     * @return .
     */
    public static JMenu createJMenu(final String title, final String
     shortcut,
        final String icon) {
        JMenu menu = createJMenu(title, shortcut);
        menu.setOpaque(true);
        menu.setIcon(ResourceLoader.get(icon));
        return menu;
    }

    /**
     *
     * @param title .
     * @param shortcut .
     * @return .
     */
    public static JMenu createJMenu(final String title, final String
     shortcut) {

        /**
         *
         * @author <a href="mailto:flu@informatik.uni-kiel.de">
         * Florian Luepke </a>
         */
        class MyMenu extends JMenu implements Preferences.Listener {

            /**
             *
             *
             */
            MyMenu() {
                setOpaque(false);
                Preferences.addListener(this);
                preferencesChanged();
            }

            /**
             *
             * @see kiel.editor.resources.Preferences.Listener#
             * preferencesChanged()
             */
            public void preferencesChanged() {
                setText(ResourceBundle.getString(title));
                if (shortcut != null) {
                    setMnemonic(ResourceBundle.getString(shortcut).charAt(0)
 );
                }
            }
        }
        return new MyMenu();
    }

    /**
     *
     * @param items .
     * @return .
     */
    public static JComboBox createJComboBox(final String[] items) {
        return new JComboBox(items) {
            {
                setOpaque(false);
                for (int i = 0; i < getComponentCount(); i++) {
                    if (getComponent(i) instanceof JComponent) {
                        ((JComponent) getComponent(i)).setOpaque(false);
                    }
                }
                ((JComponent) getRenderer()).setOpaque(false);
            }
        };
    }

    /**
     *
     * @return .
     */
    public static JSlider createJSlider() {
        return new JSlider() {
            {
                setOpaque(false);
                for (int i = 0; i < getComponentCount(); i++) {
                    if (getComponent(i) instanceof JComponent) {
                        ((JComponent) getComponent(i)).setOpaque(false);
                    }
                  }
            }
        };
    }

    /**
     *
     * @return .
     */
    public static JToolBar createJToolBar() {
        return new JToolBar() {
            protected void paintComponent(final Graphics g) {
                Utils.paintComponent(this, g, true);
            }
```

```java
        };
    }

    /**
     *
     * @param topToBottom .
     * @return .
     */
    public static JToolBar createJToolBar(final boolean topToBottom) {
        return new JToolBar() {
            protected void paintComponent(final Graphics g) {
                Utils.paintComponent(this, g, topToBottom);
            }
        };
    }

    /**
     *
     * @param layout .
     * @return .
     */
    public static JPanel createJPanel(final LayoutManager layout) {
        return new JPanel(layout) {
            protected void paintComponent(final Graphics g) {
                Utils.paintComponent(this, g, true);
            }
        };
    }

    /**
     *
     * @param layout .
     * @param topToBottom .
     * @return .
     */
    public static JPanel createJPanel(final LayoutManager layout,
        final boolean topToBottom) {
        return new JPanel(layout) {
            protected void paintComponent(final Graphics g) {
                Utils.paintComponent(this, g, topToBottom);
            }
        };
    }

    /**
     *
     * @return .
     */
    public static JLabel createJLabel() {
        return new JLabel() {
            protected void paintComponent(final Graphics g) {
                Utils.paintComponent(this, g, true);
                super.paintComponent(g);
            }
        };
    }

    /**
     *
     * @return .
     */
    public static JLabel createJLabelHorizontal() {
        return new JLabel() {
            protected void paintComponent(final Graphics g) {
                Utils.paintComponent(this, g, false);
                super.paintComponent(g);
            }
        };
    }

    /**
     *
     * @param action .
     * @param tooltip .
     * @return .
     */
    public static JButton createJButton(final Action action,
        final String tooltip) {

        /**
         *
         * @author <a href="mailto:flu@informatik.uni-kiel.de">
         * Florian Luepke </a>
         */
        final class MyJButton extends JButton implements Preferences.
 Listener {

            /**
             *
             * @param action .
             */
            MyJButton(final Action action) {
                super(action);
                setOpaque(false);
                setFocusPainted(false);
                Preferences.addListener(this);
                preferencesChanged();
            }

            /**
             *
             * @see javax.swing.AbstractButton#setText(java.lang.String)
             */
            public void setText(final String text) { }

            /**
             *
```

```
             * @see javax.swing.AbstractButton#setUI(javax.swing.plaf.
ButtonUI)
             */
            public void setUI(final ButtonUI ui) {
                super.setUI(new MetalButtonUI() {

                    /**
                     *
                     * @see javax.swing.plaf.basic.BasicButtonUI#
                     * paintButtonPressed(java.awt.Graphics,
                     * javax.swing.AbstractButton)
                     */
                    protected void paintButtonPressed(final Graphics g,
                        final AbstractButton b) {

                        Utils.paintComponent(MyJButton.this, g, true,
                            Preferences.getGuiButtonColorSelectedLight(),
                            Preferences.getGuiButtonColorSelectedDark());
                    }
                });
            }

            /**
             *
             * @see kiel.editor.resources.Preferences.Listener#
             * preferencesChanged()
             */
            public void preferencesChanged() {
                setToolTipText(ResourceBundle.getString(tooltip));
            }
        }
        return new MyJButton(action);
    }

    /**
     *
     * @param action .
     * @param tooltip .
     * @return .
     */
    public static JToggleButton createJToggleButton(final Action action,
        final String tooltip) {

        /**
         *
         * @author <a href="mailto:flu@informatik.uni-kiel.de">
         * Florian Luepke </a>
         */
        final class MyJToggleButton extends JToggleButton
            implements Preferences.Listener {

            /**
             *
             * @param action .
             */
            MyJToggleButton(final Action action) {
                super(action);
                setOpaque(false);
                setFocusPainted(false);
                Preferences.addListener(this);
                preferencesChanged();
            }

            /**
             *
             * @see javax.swing.AbstractButton#setText(java.lang.String)
             */
            public void setText(final String text) { }

            /**
             *
             * @see javax.swing.AbstractButton#setUI(javax.swing.plaf.
ButtonUI)
             */
            public void setUI(final ButtonUI ui) {

                super.setUI(new MetalToggleButtonUI() {
                    /**
                     *
                     * @see javax.swing.plaf.basic.BasicButtonUI#
                     * paintButtonPressed(java.awt.Graphics,
                     * javax.swing.AbstractButton)
                     */
                    protected void paintButtonPressed(final Graphics g,
                            final AbstractButton b) {

                        Utils.paintComponent(MyJToggleButton.this, g, true,
                            Preferences.getGuiButtonColorSelectedLight(),
                            Preferences.getGuiButtonColorSelectedDark());
                    }
                });
            }

            /**
             *
             * @see kiel.editor.resources.Preferences.Listener#
             * preferencesChanged()
             */
            public void preferencesChanged() {
                setToolTipText(ResourceBundle.getString(tooltip));
            }
        }
        return new MyJToggleButton(action);
    }

    /**
     *
     * @return .
```

```java
 */
public static JMenuBar createJMenuBar() {
    return new JMenuBar() {
        protected void paintComponent(final Graphics g) {
            Utils.paintComponent(this, g, false);
        }
    };
}

/**
 *
 * @param view .
 * @return .
 */
public static JScrollPane createJScrollPane(final Component view) {
    return new MyJScrollPane(view);
}

/**
 *
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
 a>
 */
public static class MyJScrollPane extends JScrollPane {

    /**
     * A completely non opaque component.
     * @param view .
     */
    MyJScrollPane(final Component view) {
        super(view);
        setOpaque(false);
        for (int i = 0; i < getComponentCount(); i++) {
            if (getComponent(i) instanceof JComponent) {
                ((JComponent) getComponent(i)).setOpaque(false);
            }
        }
    }

    /**
     *
     * @author <a href="mailto:flu@informatik.uni-kiel.de">
     * Florian Luepke </a>
     */
    final class MyScrollBar extends JScrollPane.ScrollBar {

        /**
         *
         * @param orientation .
         */
        MyScrollBar(final int orientation) {
            super(orientation);
            setOpaque(false);
        }

        /**
         *
         * @see javax.swing.JComponent#paintComponent(java.awt.Graphics)
         */
        protected void paintComponent(final Graphics g) {
            Utils.paintComponent(
                this,
                g,
                orientation == JScrollBar.HORIZONTAL,
                Preferences.getGuiColorLight(),
                Preferences.getGuiColorLight());
            super.paintComponent(g);
        }
    }

    /**
     *
     * @see javax.swing.JScrollPane#createHorizontalScrollBar()
     */
    public final JScrollBar createHorizontalScrollBar() {
        return new MyScrollBar(JScrollBar.HORIZONTAL);
    }

    /**
     *
     * @see javax.swing.JScrollPane#createVerticalScrollBar()
     */
    public final JScrollBar createVerticalScrollBar() {
        return new MyScrollBar(JScrollBar.VERTICAL);
    }
}

/**
 *
 * @param source .
 * @return .
 */
public static JComponent createFloatableComponentForTabbedPane(
    final JComponent source) {

    final JToolBar toolbar = Utils.createJToolBar(false);
    toolbar.setUI(new MetalToolBarUI() {

        private JTabbedPane myParent;
        private String tabTitle;
        private int tabIndex;

        protected MouseInputListener createDockingListener() {
            return new DockingListener(toolBar) {
                public void mousePressed(final MouseEvent e) {

                    super.mousePressed(e);
                    if (toolbar.getParent() instanceof JTabbedPane) {
```

```java
                            myParent = (JTabbedPane) toolbar.getParent();
                        }
                        for (int i = 0; myParent != null
                                    && i < myParent.getTabCount(); i++) {
                            if (myParent.getComponentAt(i) == toolbar) {
                                tabTitle = myParent.getTitleAt(i);
                                tabIndex = i;
                            }
                        }
                    }
                    public void mouseReleased(final MouseEvent e) {
                        super.mouseReleased(e);
                        Container parent = toolbar.getParent();
                        while (!(parent instanceof JDialog)
                            && parent.getParent() != null) {

                            parent = parent.getParent();
                        }
                        if (parent instanceof JDialog) {
                            ((JDialog) parent).setTitle(tabTitle);
                            ((JDialog) parent).setResizable(true);
                        }
                    }
                };
            }
            protected WindowListener createFrameListener() {
                return new FrameListener() {
                    public void windowClosing(final WindowEvent w) {
                        super.windowClosing(w);
                        if (tabIndex > myParent.getTabCount()) {
                            tabIndex = myParent.getTabCount();
                        }
                        myParent.add(toolbar, tabTitle, tabIndex);
                    }
                };
            }
        });
        toolbar.setLayout(new BorderLayout());
        toolbar.add(source, "Center");
        return toolbar;
    }

    /**
     *
     * @param source .
     * @return .
     */
    public static JComponent createFloatableComponentForSplitPane(
        final JComponent source) {

        final JPanel root = new JPanel(new BorderLayout());
        final JToolBar toolbar = Utils.createJToolBar(false);
        toolbar.setUI(new MetalToolBarUI() {
            private JSplitPane myParent;
            private int lastDividerLocation;

            protected MouseInputListener createDockingListener() {
                return new DockingListener(toolBar) {
                    private boolean isFirstDragEvent = true;
                    public void mouseDragged(final MouseEvent e) {
                        super.mouseDragged(e);
                        if (isFirstDragEvent) {
                            myParent = (JSplitPane) root.getParent();
                            if (myParent.getDividerLocation() > 0) {
                                lastDividerLocation =
                                    myParent.getDividerLocation();
                            }
                            if (myParent.getTopComponent() == root) {
                                myParent.setDividerLocation(0);
                            } else {
                                myParent.setDividerLocation(1.0);
                            }
                            isFirstDragEvent = false;
                        }
                    }
                    public void mouseReleased(final MouseEvent e) {
                        super.mouseReleased(e);
                        Container parent = toolbar.getParent();
                        while (!(parent instanceof JDialog)
                            && parent.getParent() != null) {

                            parent = parent.getParent();
                        }
                        if (parent instanceof JDialog) {
                            ((JDialog) parent).setResizable(true);
                            isFirstDragEvent = true;
                        }
                    }
                };
            }
            protected WindowListener createFrameListener() {
                return new FrameListener() {
                    public void windowClosing(final WindowEvent w) {
                        super.windowClosing(w);
                        myParent.setDividerLocation(lastDividerLocation);
                    }
                };
            }
        });
        toolbar.setLayout(new BorderLayout());
        root.add(toolbar, "Center");
        toolbar.add(source, "Center");

        if (source instanceof JTabbedPane) {
            final JTabbedPane tabPane = (JTabbedPane) source;
            // when the last tab is removed, then the tab pane should become
            // invisible and vice versa.
```

```
        tabPane.addContainerListener(new ContainerListener() {

            private JSplitPane parent;
            private int lastDividerLocation;

            public void componentAdded(final ContainerEvent e) {
                if (tabPane.getTabCount() == 1) {
                    if (parent != null) {
                        parent.setDividerLocation(lastDividerLocation);
                    }
                }
            }

            public void componentRemoved(final ContainerEvent e) {
                if (tabPane.getParent() != null
                    && tabPane.getParent().getParent() != null) {

                    parent = (JSplitPane) tabPane.getParent().getParent
()

                        .getParent();
                }
                if (parent != null) {
                    lastDividerLocation = parent.getDividerLocation();
                }
                if (tabPane.getTabCount() == 0) {
                    if (parent.getTopComponent()
                            == tabPane.getParent().getParent()) {
                        parent.setDividerLocation(0);
                    } else {
                        parent.setDividerLocation(1.0);
                    }
                }
            }
        });
    }
    return root;
}

/**
 * Translates the values returned by the current Resource bundle to the
 * correct accelerator.
 * @param mi the menu item to which an accelerator should be attached.
 * @return a new menu item object.
 */
public static JMenuItem addAccelerator(final JMenuItem mi) {

    String accelerator = ResourceBundle.getString("acceleratorFor"
        + mi.getAction().getClass().getName().substring(
            mi.getAction().getClass().getName().lastIndexOf('.') + 1));

    final int fKeyMinLength = 3;
    final int fKeyMaxLength = 4;
    if (!accelerator.startsWith("acceleratorFor")) {
        int c = accelerator.charAt(1);
        if (c == '+') {
            c = KeyEvent.VK_PLUS;
        } else if (c == 'F' && accelerator.length() >= fKeyMinLength
            && accelerator.length() <= fKeyMaxLength) {

            c = KeyEvent.VK_F1
                + Integer.parseInt(accelerator.substring(2)) - 1;
        } else if (accelerator.equals("ODEL")) {
            c = KeyEvent.VK_DELETE;
        }
        mi.setAccelerator(KeyStroke.getKeyStroke(c,
            Integer.parseInt(accelerator.substring(0, 1))));
    }
    return mi;
}

/**
 * Convinience method for setting an accelerator.
 * @param actionClass for which action a menu item should be created.
 * @return a new menu item.
 */
public static JMenuItem createJMenuItem(final Class actionClass) {
    return addAccelerator(new JMenuItem(EditorActions.get(
        actionClass)));
}
}
```

kiel.editor.controller.ActionAddLoopTransition

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import org.jgraph.graph.DefaultGraphCell;
import org.jgraph.graph.Port;

/**
 * Defines the edit step "add loop transition".
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.22 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ActionAddLoopTransition extends EditorAction {

    /**
     * Creates an object with the connect.gif image.
     */
    ActionAddLoopTransition() {
        super("connect.gif");
    }

    /**
     * Searches in the selected cell for the first port child.
     *
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().getUndoClusterManager().setUndoClusterStart()
 ;
        DefaultGraphCell cell =
            (DefaultGraphCell) getEditor().getGraph().getSelectionCell();
        if (cell != null) {
            for (int i = 0; i < cell.getChildCount(); i++) {
                if (cell.getChildAt(i) instanceof Port) {
                    try {
                        getEditor().getGraph().getMyGraphModel().
 addTransition(
                            (Port) cell.getChildAt(i),
                            (Port) cell.getChildAt(i),
                            getEditor().getCurrentTransitionType(),
                            getEditor().getCurrentTransitionLine(),
                            getEditor().getGraph().getCurrentPopupPoint());
                    } catch (Exception ex) {
                        getEditor().getKielFrame().handleException(
                            ex.getMessage(), false, ex);
                    }
                    break;
                }
            }
        }
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
 this);
    }
}
```

kiel.editor.controller.ActionAddTransitionToggle

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.resources.ResourceLoader;

/**
 * Defines the edit step "toggle transition mode on/off".
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.15 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ActionAddTransitionToggle extends EditorAction {

    /**
     * Creates an object with an connecton.gif image.
     */
    ActionAddTransitionToggle() {
        super("connecton.gif");
    }

    /**
     * Toggles the image and calls JGraph.setPortsVisible().
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().setPortsVisible(
            !getEditor().getGraph().isPortsVisible());
        if (getEditor().getGraph().isPortsVisible()) {
            putValue(SMALL_ICON, ResourceLoader.get("connecton.gif"));
        } else {
            putValue(SMALL_ICON, ResourceLoader.get("connectoff.gif"));
        }
    }
}
```

kiel.editor.controller.ActionEditCollapsed

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;
import java.awt.geom.Rectangle2D;

import org.jgraph.graph.DefaultGraphCell;
import org.jgraph.graph.GraphConstants;

import kiel.editor.MorphingLayouter;
import kiel.editor.graph.CompositeStateCell;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.graph.RegionCell;
import kiel.editor.resources.ResourceBundle;

/**
 * Defines the edit step "collapse" for collapsing or expanding the
     currently
 * selected node.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.26 $ last modified $Date: 2005/08/01 17:53:17 $
 */
public final class ActionEditCollapsed extends EditorAction
    implements MorphingLayouter.MyListener {

    /**
     * This action does not have any image since it will be represented by
     * a checkbox.
     */
    ActionEditCollapsed() {
        super(null);
    }

    /**
     * Calls MyGraphModel.flipCollapsed().
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {

        getEditor().getGraph().getUndoClusterManager().setUndoClusterStart()
     ;
        DefaultGraphCell selected =
            (DefaultGraphCell) getEditor().getGraph().getSelectionCell();

        if (selected instanceof CompositeStateCell) {

            boolean isCollapsed = MyGraphConstants.isCollapsed(
                selected.getAttributes());

            // check if enough space for expanding is available...
            if (isCollapsed) {

                Rectangle2D parentsBounds =
                    GraphConstants.getBounds(
                        ((DefaultGraphCell) selected.getParent())
                        .getAttributes());
                Rectangle2D selectedBounds =
                    GraphConstants.getBounds(
                        selected.getAttributes());
                Rectangle2D expandedBounds =
                    MyGraphConstants.getExpandedBounds(
                        selected.getAttributes()).getBounds2D();
                expandedBounds.setRect(
                    selectedBounds.getX(),
                    selectedBounds.getY(),
                    expandedBounds.getWidth(),
                    expandedBounds.getHeight());
                if ((selected.getParent() instanceof RegionCell
                        && !parentsBounds.contains(expandedBounds))
                        // intersect or contain...
                  || getEditor().getGraph().getMyGraphModel().intersect(
                        selected, expandedBounds, true, null, true)) {

                    getEditor().getKielFrame().handleException(
                        ResourceBundle.getString("notEnoughSpace"),
                        false, null);
                    return;
                }
            }
            getEditor().getGraph().getMyGraphModel().flipCollapsed(
                getEditor().getGraph(),
                (CompositeStateCell) getEditor().getGraph().getSelectionCell
 (),
                getEditor().getGraph().getGraphLayoutCache(), this);
        }
    }

    /**
     * Terminates the current undo cluster.
     * @see kiel.editor.MorphingLayouter.MyListener#morphingFinished()
     */
    public void morphingFinished() {
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
     this);
    }
}
```

kiel.editor.controller.ActionEditConvertFlipFinalAttribute

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.graph.ANDStateCell;
import kiel.editor.graph.FinalANDStateCell;
import kiel.editor.graph.FinalORStateCell;
import kiel.editor.graph.FinalSimpleStateCell;
import kiel.editor.graph.ORStateCell;
import kiel.editor.graph.SimpleStateCell;

import org.jgraph.graph.DefaultGraphCell;

/**
 * Converts the currently selected node to a FinalState. If it is an ORState
     , it
 * will become an FinalORState, if it was a SimpleState, it will become an
 * FinalSimpleState.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.1 $ last modified $Date: 2005/04/27 17:00:34 $
 */
public final class ActionEditConvertFlipFinalAttribute extends EditorAction
     {

    /**
     * Creates an object with an FinalSimpleState.gif image.
     */
    ActionEditConvertFlipFinalAttribute() {
        super("FinalSimpleState.gif");
    }

    /**
     * Calls MyGraphModel.convert().
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().getUndoClusterManager().setUndoClusterStart()
     ;
        DefaultGraphCell selectedCell =
            (DefaultGraphCell) getEditor().getGraph().getSelectionCell();
        if (selectedCell instanceof FinalORStateCell) {
            getEditor().getGraph().getMyGraphModel().convert(
                selectedCell, ORStateCell.class);
        } else if (selectedCell instanceof FinalANDStateCell) {
            getEditor().getGraph().getMyGraphModel().convert(
                selectedCell, ANDStateCell.class);
        } else if (selectedCell instanceof FinalSimpleStateCell) {
            getEditor().getGraph().getMyGraphModel().convert(
                selectedCell, SimpleStateCell.class);
        } else if (selectedCell instanceof ORStateCell) {
            getEditor().getGraph().getMyGraphModel().convert(
                selectedCell, FinalORStateCell.class);
        } else if (selectedCell instanceof ANDStateCell) {
            getEditor().getGraph().getMyGraphModel().convert(
                selectedCell, FinalANDStateCell.class);
        } else if (selectedCell instanceof SimpleStateCell) {
            getEditor().getGraph().getMyGraphModel().convert(
                selectedCell, FinalSimpleStateCell.class);
        }
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
     this);
    }
}
```

kiel.editor.controller.ActionEditConvertToBezier

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import org.jgraph.graph.GraphCell;
import org.jgraph.graph.GraphConstants;

/**
 * Converts the currently selected transition to a bezier curve.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.16 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionEditConvertToBezier extends EditorAction {

    /**
     * Creates an object with an edge.gif image.
     */
    ActionEditConvertToBezier() {
        super("edge.gif");
    }

    /**
     * Calls MyGraphModel.changeLineStyle().
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().getUndoClusterManager().setUndoClusterStart()
     ;
        getEditor().getGraph().getMyGraphModel().changeLineStyle(
            (GraphCell) getEditor().getGraph().getSelectionCell(),
            GraphConstants.STYLE_BEZIER);
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
     this);
    }
}
```

kiel.editor.controller.ActionEditConvertToNormalTransition

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.graph.NormalTerminationCell;

import org.jgraph.graph.DefaultGraphCell;

/**
 * Converts the currently selected transition to a NormalTransition.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.16 $ last modified $Date: 2005/03/08 15:49:19 $
 */
public final class ActionEditConvertToNormalTransition extends EditorAction
     {

    /**
     * Creates an object with an arrow.gif image.
     */
    ActionEditConvertToNormalTransition() {
        super("arrow.gif");
    }

    /**
     * Calls MyGraphModel.convert().
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().getUndoClusterManager().setUndoClusterStart()
     ;
        getEditor().getGraph().getMyGraphModel().convert(
            (DefaultGraphCell) getEditor().getGraph().getSelectionCell(),
            NormalTerminationCell.class);
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
     this);
    }
}
```

kiel.editor.controller.ActionEditConvertToORState

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.graph.ORStateCell;

import org.jgraph.graph.DefaultGraphCell;

/**
 * Converts the currently selected node to an ORState.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.16 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionEditConvertToORState extends EditorAction {

    /**
     * Creates an object with an OrState.gif image.
     */
    ActionEditConvertToORState() {
        super("OrState.gif");
    }

    /**
     * Calls MyGraphModel.convert().
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().getUndoClusterManager().setUndoClusterStart()
     ;
        getEditor().getGraph().getMyGraphModel().convert(
            (DefaultGraphCell) getEditor().getGraph().getSelectionCell(),
            ORStateCell.class);
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
     this);
    }
}
```

kiel.editor.controller.ActionEditConvertToOrthogonal

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import org.jgraph.graph.GraphCell;
import org.jgraph.graph.GraphConstants;

/**
 * Converts the currently selected transition to an orthogonal type.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.16 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionEditConvertToOrthogonal extends EditorAction {

    /**
     * Creates an object with an edge.gif image.
     */
    ActionEditConvertToOrthogonal() {
        super("edge.gif");
    }

    /**
     * Calls MyGraphModel.changeLineStyle().
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().getUndoClusterManager().setUndoClusterStart()
    ;
        getEditor().getGraph().getMyGraphModel().changeLineStyle(
            (GraphCell) getEditor().getGraph().getSelectionCell(),
            GraphConstants.STYLE_ORTHOGONAL);
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
     this);
    }
}
```

kiel.editor.controller.ActionEditConvertToSimpleState

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.graph.SimpleStateCell;

import org.jgraph.graph.DefaultGraphCell;

/**
 * Converts the currently selected node to an SimpleState.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.1 $ last modified $Date: 2005/04/01 23:56:02 $
 */
public final class ActionEditConvertToSimpleState extends EditorAction {

    /**
     * Creates an object with an SimpleState.gif image.
     */
    ActionEditConvertToSimpleState() {
        super("SimpleState.gif");
    }

    /**
     * Calls MyGraphModel.convert().
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().getUndoClusterManager().setUndoClusterStart()
 ;
        getEditor().getGraph().getMyGraphModel().convert(
            (DefaultGraphCell) getEditor().getGraph().getSelectionCell(),
            SimpleStateCell.class);
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
 this);
    }
}
```

kiel.editor.controller.ActionEditConvertToSpline

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import org.jgraph.graph.GraphCell;
import org.jgraph.graph.GraphConstants;

/**
 * Converts the currently selected transition to an orthogonal type.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.16 $ last modified $Date: 2005/03/08 15:49:19 $
 */
public final class ActionEditConvertToSpline extends EditorAction {

    /**
     * Creates an object with an edge.gif image.
     */
    ActionEditConvertToSpline() {
        super("edge.gif");
    }

    /**
     * Calls MyGraphModel.changeLineStyle().
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().getUndoClusterManager().setUndoClusterStart()
 ;
        getEditor().getGraph().getMyGraphModel().changeLineStyle(
            (GraphCell) getEditor().getGraph().getSelectionCell(),
            GraphConstants.STYLE_SPLINE);
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
 this);
    }
}
```

kiel.editor.controller.ActionEditConvertToStrongAbortion

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.graph.StrongAbortionCell;

import org.jgraph.graph.DefaultGraphCell;

/**
 * Converts the currently selected transition to a StrongAbortion.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.16 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionEditConvertToStrongAbortion extends EditorAction {

    /**
     * Creates an object with an arrow.gif image.
     */
    ActionEditConvertToStrongAbortion() {
        super("arrow.gif");
    }

    /**
     * Calls MyGraphModel.convert().
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().getUndoClusterManager().setUndoClusterStart()
    ;
        getEditor().getGraph().getMyGraphModel().convert(
            (DefaultGraphCell) getEditor().getGraph().getSelectionCell(),
            StrongAbortionCell.class);
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
    this);
    }
}
```

kiel.editor.controller.ActionEditConvertToWeak Abortion

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.graph.WeakAbortionCell;

import org.jgraph.graph.DefaultGraphCell;

/**
 * Converts the currently selected transition to a WeakAbortion.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.17 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ActionEditConvertToWeakAbortion extends EditorAction {

    /**
     * Creates an object with an arrow.gif image.
     */
    ActionEditConvertToWeakAbortion() {
        super("arrow.gif");
    }

    /**
     * Calls MyGraphModel.convert().
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().getUndoClusterManager().setUndoClusterStart()
    ;
        getEditor().getGraph().getMyGraphModel().convert(
            (DefaultGraphCell) getEditor().getGraph().getSelectionCell(),
            WeakAbortionCell.class);
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
    this);
    }
}
```

kiel.editor.controller.ActionEditCopy

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import javax.swing.Action;

import kiel.editor.resources.ResourceLoader;

/**
 * Uses the swing.TransferHandler and implements the well known
 * C&P-functionality.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionEditCopy extends EditorAction {

    /**
    * Wraps the swing.TransferHandler.CopyAction.
    */
    private Action copy;

    /**
     * Creates an object with an Copy16.gif image.
     */
    ActionEditCopy() {
        super("Copy16.gif");
        setEnabled(false);
        Action action = javax.swing.TransferHandler.getCopyAction();
        action.putValue(Action.SMALL_ICON, ResourceLoader.get("Copy16.gif"))
 ;
        copy = new EventRedirector(action);
    }

    /**
     * Nothin implemented yet.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        // TODO Implement!
    }

    /**
     * Currently permits the given value <code>true</code>.
     * @see javax.swing.Action#setEnabled(boolean)
     */
    public void setEnabled(final boolean b) {
        if (!b) {
            super.setEnabled(b);
        }
    }
}
```

kiel.editor.controller.ActionEditCut

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import javax.swing.Action;

import kiel.editor.resources.ResourceLoader;

import org.jgraph.graph.DefaultGraphCell;

/**
 * Uses the swing.TransferHandler and implements the well known
 * C&P-functionality.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.17 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ActionEditCut extends EditorAction {

    /**
     * The currently cut cell has to be stored until the paste action is
     * performed.
     */
    private DefaultGraphCell currentCutCell;

    /**
     * @see #currentCutCell.
     * @param cell .
     */
    public void setCurrentCutCell(final DefaultGraphCell cell) {
        currentCutCell = cell;
    }

    /**
     * @see #currentCutCell.
     * @return .
     */
    public DefaultGraphCell getCurrentCutCell() {
        return currentCutCell;
    }

    /**
     * Wraps the swing.TransferHandler.CutAction.
     */
    private Action cut;

    /**
     * Creates an object with an Cut16.gif image.
     */
    ActionEditCut() {
        super("Cut16.gif");
        Action action = javax.swing.TransferHandler.getCutAction();
        action.putValue(Action.SMALL_ICON, ResourceLoader.get("Cut16.gif"));
        cut = new EditorAction.EventRedirector(action);
    }

    /**
     * Calls JGraphModel.remove() and delegates the actionPerformed() to the
     * cut object.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().getUndoClusterManager()
            .setUndoClusterStart();
        currentCutCell = (DefaultGraphCell) getEditor().getGraph()
            .getSelectionCell();
        getEditor().getGraph().getModel().remove(
                new Object[]{currentCutCell});
        cut.actionPerformed(e);
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
     this);
    }
}
```

kiel.editor.controller.ActionEditEdit

```
package kiel.editor.controller;

import java.awt.GridLayout;
import java.awt.event.ActionEvent;

import javax.swing.JCheckBox;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.tree.DefaultMutableTreeNode;

import org.jgraph.graph.DefaultGraphCell;

import kiel.dataStructure.CompoundLabel;
import kiel.dataStructure.Transition;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.graph.NodeCell;
import kiel.editor.graph.TransitionCell;
import kiel.editor.resources.ResourceBundle;
import kiel.util.CompoundLabelException;

/**
 * Starts the text editor in a graph cell.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.20 $ last modified $Date: 2005/07/27 09:42:14 $
 */
public final class ActionEditEdit extends EditorAction {

    /**
     * Creates an object with an Edit16.gif image.
     */
    ActionEditEdit() {
        super("Edit16.gif");
    }

    /**
     * Store the current userObject of the selected graph cell.
     */
    private String currentUserObject;

    /**
     * Calls the JGraph.startEditingAtCell() to open a textfield on a node.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().getUndoClusterManager().setUndoClusterStart()
         ;
        currentUserObject = (String) ((DefaultMutableTreeNode) getEditor()
            .getGraph().getSelectionCell()).getUserObject();

        final DefaultGraphCell selected =
            (DefaultGraphCell) getEditor().getGraph().getSelectionCell();

        if (selected instanceof TransitionCell) {
            Transition transition = (Transition)
                MyGraphConstants.getGraphicalObject(selected.getAttributes()
);

            JTextField textFieldTrigger = new JTextField();
            JTextField textFieldCondition = new JTextField();
            JTextField textFieldEffect = new JTextField();
            JCheckBox checkBoxImmediate = new JCheckBox();

            if (transition.getLabel() instanceof CompoundLabel) {
                CompoundLabel label = (CompoundLabel) transition.getLabel();
                if (label.getCondition() != null) {
                    textFieldCondition.setText(label.getCondition() + "");
                }
                if (label.getEffect() != null) {
                    textFieldEffect.setText(label.getEffect() + "");
                }
                if (label.getTrigger() != null) {
                    checkBoxImmediate.setSelected(label.getTrigger()
                        .isImmediate());
                    if (!label.getTrigger().isImmediate()) {
                        textFieldTrigger.setText(label.getTrigger() + "");
                    } else if ((label.getTrigger() + "").length() > 2) {
                        textFieldTrigger.setText(
                            label.getTrigger().toString().substring(2));
                    }
                }
            }
            final int cols = 5;
            JPanel p = new JPanel(new GridLayout(2, cols));
            p.add(new JLabel("Immediate:   "));
            p.add(new JLabel("Trigger:   "));
            p.add(new JLabel("Condition:   "));
            p.add(new JLabel("Action:   "));
            p.add(checkBoxImmediate);
            p.add(textFieldTrigger);
            p.add(textFieldCondition);
            p.add(textFieldEffect);
            int answer = JOptionPane.showOptionDialog(
                getEditor().getKielFrame().getJFrame(),
                p, ResourceBundle.getString("editTransitionLabel"),
                JOptionPane.OK_CANCEL_OPTION,
                JOptionPane.QUESTION_MESSAGE,
                null, new Object[] {
```

```
                    ResourceBundle.getString("OK"),
                    ResourceBundle.getString("Cancel")},
                    ResourceBundle.getString("OK"));
            if (answer == 0) {
                try {
                    getEditor().getGraph().getMyGraphModel()
                        .setTransitionLabel(
                            selected,
                            textFieldTrigger.getText(),
                            textFieldCondition.getText(),
                            textFieldEffect.getText(),
                            checkBoxImmediate.isSelected(),
                            currentUserObject);
                } catch (CompoundLabelException ex) {
                    getEditor().getKielFrame().handleException(
                        ex.getMessage(), false, ex);
                }
            }
        } else {
            JTextField textFieldLabel = new JTextField(currentUserObject);

            int answer = JOptionPane.showOptionDialog(
                getEditor().getKielFrame().getJFrame(),
                textFieldLabel, ResourceBundle.getString("editNodeLabel"),
                JOptionPane.OK_CANCEL_OPTION,
                JOptionPane.QUESTION_MESSAGE,
                null, new Object[] {
                    ResourceBundle.getString("OK"),
                    ResourceBundle.getString("Cancel")},
                    ResourceBundle.getString("OK"));
            if (answer == 0) {
                getEditor().getGraph().getMyGraphModel()
                    .setNodeLabel((NodeCell) selected,
                        textFieldLabel.getText());
            }
        }
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
     this);
    }

    /**
     * Is called to terminate the current undo cluster.
     * @see kiel.editor.EditListener#editingStopped()
    public void editingStopped() {

        final DefaultGraphCell selected =
            (DefaultGraphCell) getEditor().getGraph().getSelectionCell();

        if (selected instanceof TransitionCell) {
            try {
                getEditor().getGraph().getMyGraphModel()
                    .postEditTransitionLabel(selected, currentUserObject);
            } catch (CompoundLabelException e) {
                getEditor().getKielFrame().handleException(
                    e.getMessage(), false, e);
            }
        }
        getEditor().getLogFile().log(0, "EditingStopped ActionEditEdit");
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
     this);
    }
     */
}
```

kiel.editor.controller.ActionEditFind

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

/**
 * Represents a search action, but is not implemented yet. the idea is to
 * provide a search function for viewing nodes.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.14 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ActionEditFind extends EditorAction {

    /**
     * Creates an object with an Find16.gif image.
     */
    ActionEditFind() {
        super("Find16.gif");
        setEnabled(false);
    }

    /**
     * TODO Not implemented yet.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
    }

    /**
     * Permits true value.
     * @see javax.swing.Action#setEnabled(boolean)
     */
    public void setEnabled(final boolean b) {
        if (!b) {
            super.setEnabled(b);
        }
    }
}
```

kiel.editor.controller.ActionEditPaste

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;
import java.awt.geom.Rectangle2D;

import javax.swing.Action;

import kiel.editor.resources.ResourceLoader;

import org.jgraph.graph.AttributeMap;
import org.jgraph.graph.DefaultGraphCell;
import org.jgraph.graph.GraphConstants;

/**
 * The Paste part of a Copy&Paste action.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.17 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ActionEditPaste extends EditorAction {

    /**
     * The wrapped paste action object.
     */
    private Action paste;

    /**
     * Creates an object with an Paste16.gif image.
     */
    ActionEditPaste() {
        super("Paste16.gif");
        Action action = javax.swing.TransferHandler.getPasteAction();
        action.putValue(Action.SMALL_ICON, ResourceLoader.get("Paste16.gif")
     );
        paste = new EventRedirector(action);

    }

    /**
     * Pastes in the cut node.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {

        DefaultGraphCell currentCutCell = ((ActionEditCut) EditorActions.get
     (
            ActionEditCut.class)).getCurrentCutCell();
        if (currentCutCell != null) {
            getEditor().getGraph().getUndoClusterManager()
            .setUndoClusterStart();
            paste.actionPerformed(e);
            Rectangle2D bounds =
                GraphConstants.getBounds(currentCutCell.getAttributes());
            GraphConstants.setBounds(currentCutCell
                    .getAttributes(), new AttributeMap.
     SerializableRectangle2D(
                        getEditor().getGraph().getCurrentPopupPoint().getX()
     ,
                        getEditor().getGraph().getCurrentPopupPoint().getY()
     ,
                        (int) bounds.getWidth(),
                        (int) bounds.getHeight()));
            getEditor().getGraph().getMyGraphModel().insert(
                    new Object[]{currentCutCell}, null,
                    null, null, null);
            getEditor().getGraph().getMyGraphModel().changeParent(
                currentCutCell, getEditor().getGraph().getGraphLayoutCache()
     );
            getEditor().getGraph().getUndoClusterManager()
                .setUndoClusterEnd(this);
        }
    }
}
```

kiel.editor.controller.ActionEditRedo

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;

/**
 * Represents the Redo function. Takes the last undo action from the stack
     and
 * re-executes it.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionEditRedo extends EditorAction {

    /**
     * Creates an object with an Redo16.gif image.
     */
    ActionEditRedo() {
        super("Redo16.gif");
        setEnabled(false);
    }

    /**
     * Delegates to the UndoClusterManager.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().getUndoClusterManager().redo();
    }
}
```

kiel.editor.controller.ActionEditRemove

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

/**
 * Represents the DELETE action, triggered by a menu item or by pressing the
 * DEL key.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.14 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionEditRemove extends EditorAction {

    /**
     * Creates an object with an Delete16.gif image.
     */
    ActionEditRemove() {
        super("Delete16.gif");
        setEnabled(false);
    }

    /**
     * Calls JGraph.remove(). Deletes the selected node and its children.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {

        if (!getEditor().getGraph().isSelectionEmpty()) {
            getEditor().getGraph().getUndoClusterManager()
                .setUndoClusterStart();
            Object[] cells = getEditor().getGraph().getSelectionCells();
            cells = getEditor().getGraph().getDescendants(cells);
            getEditor().getGraph().getModel().remove(cells);
            getEditor().getGraph().getUndoClusterManager()
                .setUndoClusterEnd(this);
        }
    }
}
```

kiel.editor.controller.ActionEditSelectAll

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;
import java.util.ArrayList;
import java.util.Arrays;

/**
 * Selects all children of the currently selected node and desects the
 * currently selected node.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionEditSelectAll extends EditorAction {

    /**
     * Creates an object with an group.gif image.
     */
    ActionEditSelectAll() {
        super("group.gif");
    }

    /**
     * Selects all child nodes of the selected cells.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        // TODO does not work, take care of mouse clicks destroying the
        // selection
        Object[] roots = getEditor().getGraph().getSelectionCells();
        ArrayList children = new ArrayList();
        children.addAll(
            Arrays.asList(getEditor().getGraph().getDescendants(roots)));
        for (int i = 0; i < roots.length; i++) {
            children.remove(roots[i]);
        }
        getEditor().getGraph().setSelectionCells(children.toArray());
    }
}
```

kiel.editor.controller.ActionEditStateChartInputEvents

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;
import java.util.ArrayList;

import javax.swing.JOptionPane;
import javax.swing.JTextField;

import kiel.dataStructure.StateChart;
import kiel.dataStructure.eventexp.Event;
import kiel.editor.resources.ResourceBundle;

/**
 * Shows an edit dialog to change the state's events.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.5 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ActionEditStateChartInputEvents extends EditorAction {

    /**
     * Creates an object with an Edit16.gif image.
     */
    ActionEditStateChartInputEvents() {
        super("Edit16.gif");
    }

    /**
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        StateChart statechart = getEditor().getGraph().getMyGraphModel()
            .getCurrentStateChart();
        ArrayList events = statechart.getInputEvents();
        String eventsText = events.toString();
        eventsText = eventsText.substring(1, eventsText.length() - 1);

        final int columns = 50;
        JTextField textField = new JTextField(eventsText, columns);
        int answer = JOptionPane.showOptionDialog(
            getEditor().getKielFrame().getJFrame(),
            textField,
            ResourceBundle.getString("editStatechartInputEvents"),
            JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null, new Object[] {
                ResourceBundle.getString("OK"),
                ResourceBundle.getString("Cancel")},
                ResourceBundle.getString("OK"));
        if (answer == 0) {
            try {
                statechart.setInputEvents((Event[]) getEditor().getKielFrame
()

                    .parseSignals(textField.getText()).toArray(new Event[0])
);
                getEditor().getGraph().repaint();
                getEditor().getLogFile().log(0,
                    "Parsed events: " + statechart.getInputEvents());
            } catch (Exception ex) {
                getEditor().getKielFrame().handleException(
                    ex.getMessage(), false, ex);
            }
        }
    }
}
```

kiel.editor.controller.ActionEditStateChartOutputEvents

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;
import java.util.ArrayList;

import javax.swing.JOptionPane;
import javax.swing.JTextField;

import kiel.dataStructure.StateChart;
import kiel.dataStructure.eventexp.Event;
import kiel.editor.resources.ResourceBundle;

/**
 * Shows an edit dialog to change the state's events.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.5 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ActionEditStateChartOutputEvents extends EditorAction {

    /**
     * Creates an object with an Edit16.gif image.
     */
    ActionEditStateChartOutputEvents() {
        super("Edit16.gif");
    }

    /**
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        StateChart statechart = getEditor().getGraph().getMyGraphModel()
            .getCurrentStateChart();
        ArrayList events = statechart.getOutputEvents();
        String eventsText = events.toString();
        eventsText = eventsText.substring(1, eventsText.length() - 1);

        final int columns = 50;
        JTextField textField = new JTextField(eventsText, columns);
        int answer = JOptionPane.showOptionDialog(
            getEditor().getKielFrame().getJFrame(),
            textField,
            ResourceBundle.getString("editStatechartOutputEvents"),
            JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null, new Object[] {
                ResourceBundle.getString("OK"),
                ResourceBundle.getString("Cancel")},
                ResourceBundle.getString("OK"));
        if (answer == 0) {
            try {
                statechart.setOutputEvents((Event[]) getEditor().
getKielFrame()
                    .parseSignals(textField.getText()).toArray(new Event[0])
);
                getEditor().getGraph().repaint();
                getEditor().getLogFile().log(0,
                    "Parsed events: " + statechart.getOutputEvents());
            } catch (Exception ex) {
                getEditor().getKielFrame().handleException(
                    ex.getMessage(), false, ex);
            }
        }
    }
}
```

kiel.editor.controller.ActionEditStateDoActions

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import javax.swing.JOptionPane;
import javax.swing.JTextField;

import org.jgraph.graph.DefaultGraphCell;

import kiel.dataStructure.CompositeState;
import kiel.dataStructure.State;
import kiel.dataStructure.action.Actions;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.resources.ResourceBundle;
import kiel.util.CompoundLabelException;
import kiel.util.CompoundLabelParser;

/**
 * Shows an edit dialog to change the state's do actions.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.15 $ last modified $Date: 2005/07/14 15:27:55 $
 */
public final class ActionEditStateDoActions extends EditorAction {

    /**
     * Creates an object with an Edit16.gif image.
     */
    ActionEditStateDoActions() {
        super("Edit16.gif");
    }

    /**
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {

        State state = (State) MyGraphConstants
            .getGraphicalObject(((DefaultGraphCell) getEditor().getGraph()
                .getSelectionCell()).getAttributes());
        String actionsText = state.getDoActivity().toString();

        final int columns = 50;
        JTextField textfield = new JTextField(actionsText, columns);
        int answer = JOptionPane.showOptionDialog(
            getEditor().getKielFrame().getJFrame(),
            textfield,
            ResourceBundle.getString("EditStateDoActivities"),
            JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null, new Object[] {
                ResourceBundle.getString("OK"),
                ResourceBundle.getString("Cancel")},
                ResourceBundle.getString("OK"));
        if (answer == 0) {
            try {
                CompoundLabelParser.setStateChart(
                    getEditor().getCurrentStateChart());
                if (state.getParent() != null) {
                    CompoundLabelParser.setCompositeState(state.getParent())
;
                } else {
                    CompoundLabelParser.setCompositeState(
                        (CompositeState) state);
                }
                if (textfield.getText().trim().length() == 0) {
                    state.setDoActivity(new Actions());
                } else {
                    state.setDoActivity(
                        CompoundLabelParser.parseActions(textfield.getText()
));
                }
                getEditor().getGraph().repaint();
            } catch (CompoundLabelException ex) {
                getEditor().getKielFrame().handleException(
                    ex.getMessage(), false, ex);
            }
        }
    }
}
```

kiel.editor.controller.ActionEditStateEntryActions

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import javax.swing.JOptionPane;
import javax.swing.JTextField;

import org.jgraph.graph.DefaultGraphCell;

import kiel.dataStructure.CompositeState;
import kiel.dataStructure.State;
import kiel.dataStructure.action.Actions;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.resources.ResourceBundle;
import kiel.util.CompoundLabelException;
import kiel.util.CompoundLabelParser;

/**
 * Shows an edit dialog to change the state's entry actions.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.15 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ActionEditStateEntryActions extends EditorAction {

    /**
     * Creates an object with an Edit16.gif image.
     */
    ActionEditStateEntryActions() {
        super("Edit16.gif");
    }

    /**
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {

        State state = (State) MyGraphConstants
            .getGraphicalObject(((DefaultGraphCell) getEditor().getGraph()
                .getSelectionCell()).getAttributes());
        String actionsText = state.getEntry().toString();

        final int columns = 50;
        JTextField textfield = new JTextField(actionsText, columns);
        int answer = JOptionPane.showOptionDialog(
            getEditor().getKielFrame().getJFrame(),
            textfield,
            ResourceBundle.getString("editOnEntryActions"),
            JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null, new Object[] {
                ResourceBundle.getString("OK"),
                ResourceBundle.getString("Cancel")},
                ResourceBundle.getString("OK"));
        if (answer == 0) {
            try {
                CompoundLabelParser.setStateChart(
                    getEditor().getCurrentStateChart());
                if (state.getParent() != null) {
                    CompoundLabelParser.setCompositeState(state.getParent())
;
                } else {
                    CompoundLabelParser.setCompositeState(
                        (CompositeState) state);
                }
                if (textfield.getText().trim().length() == 0) {
                    state.setEntry(new Actions());
                } else {
                    state.setEntry(
                        CompoundLabelParser.parseActions(textfield.getText()
));
                }
                getEditor().getGraph().repaint();
            } catch (CompoundLabelException ex) {
                getEditor().getKielFrame().handleException(
                    ex.getMessage(), false, ex);
            }
        }
    }
}
```

kiel.editor.controller.ActionEditStateEvents

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;
import java.util.Collection;

import javax.swing.JOptionPane;
import javax.swing.JTextField;

import org.jgraph.graph.DefaultGraphCell;

import kiel.dataStructure.CompositeState;
import kiel.dataStructure.eventexp.Event;
import kiel.dataStructure.eventexp.IntegerSignal;
import kiel.dataStructure.intexp.IntegerExpression;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.resources.ResourceBundle;

/**
 * Shows an edit dialog to change the state's events.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.16 $ last modified $Date: 2005/07/21 16:47:53 $
 */
public final class ActionEditStateEvents extends EditorAction {

    /**
     * Creates an object with an Edit16.gif image.
     */
    ActionEditStateEvents() {
        super("Edit16.gif");
    }

    /**
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {

        CompositeState state = (CompositeState) MyGraphConstants
            .getGraphicalObject(((DefaultGraphCell) getEditor().getGraph()
                .getSelectionCell()).getAttributes());

        String eventsText = "";
        for (int i = 0; i < state.getLocalEvents().size(); i++) {
            Event event = (Event) state.getLocalEvents().get(i);
            eventsText += event.getName();

            if (event instanceof IntegerSignal) {
                IntegerExpression expr =
                    ((IntegerSignal) event).getInitialValue();
                if (expr != null) {
                    eventsText += " := " + expr.toIntExpString();
                }
                eventsText += " : integer";
            }
            if (i < state.getLocalEvents().size() - 1) {
                eventsText += ", ";
            }
        }

        final int columns = 50;
        JTextField textField = new JTextField(eventsText, columns);
        int answer = JOptionPane.showOptionDialog(
            getEditor().getKielFrame().getJFrame(),
            textField,
            ResourceBundle.getString("EditStatechartEvents"),
            JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null, new Object[] {
                ResourceBundle.getString("OK"),
                ResourceBundle.getString("Cancel")},
                ResourceBundle.getString("OK"));
        if (answer == 0) {
            try {
                Collection newLocalEvents = getEditor().getKielFrame()
                    .parseSignals(textField.getText());
                state.getLocalEvents().clear();
                state.addLocalEvents(newLocalEvents);
                getEditor().getGraph().repaint();
                getEditor().getLogFile().log(0,
                    "Parsed events: " + newLocalEvents);
            } catch (Exception ex) {
                getEditor().getKielFrame().handleException(
                    ex.getMessage(), false, ex);
            }
        }
    }
}
```

kiel.editor.controller.ActionEditStateExitActions

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import javax.swing.JOptionPane;
import javax.swing.JTextField;

import org.jgraph.graph.DefaultGraphCell;

import kiel.dataStructure.CompositeState;
import kiel.dataStructure.State;
import kiel.dataStructure.action.Actions;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.resources.ResourceBundle;
import kiel.util.CompoundLabelException;
import kiel.util.CompoundLabelParser;

/**
 * Shows an edit dialog to change the state's exit actions.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.15 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ActionEditStateExitActions extends EditorAction {

    /**
     * Creates an object with an Edit16.gif image.
     */
    ActionEditStateExitActions() {
        super("Edit16.gif");
    }

    /**
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {

        State state = (State) MyGraphConstants
            .getGraphicalObject(((DefaultGraphCell) getEditor().getGraph()
                .getSelectionCell()).getAttributes());
        String actionsText = state.getExit().toString();

        final int columns = 50;
        JTextField textfield = new JTextField(actionsText, columns);
        int answer = JOptionPane.showOptionDialog(
            getEditor().getKielFrame().getJFrame(),
            textfield,
            ResourceBundle.getString("editOnExitActions"),
            JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null, new Object[] {
                ResourceBundle.getString("OK"),
                ResourceBundle.getString("Cancel")},
                ResourceBundle.getString("OK"));
        if (answer == 0) {
            try {
                CompoundLabelParser.setStateChart(
                    getEditor().getCurrentStateChart());
                if (state.getParent() != null) {
                    CompoundLabelParser.setCompositeState(state.getParent())
;
                } else {
                    CompoundLabelParser.setCompositeState(
                        (CompositeState) state);
                }
                if (textfield.getText().trim().length() == 0) {
                    state.setExit(new Actions());
                } else {
                    state.setExit(
                        CompoundLabelParser.parseActions(textfield.getText()
));
                }
                getEditor().getGraph().repaint();
            } catch (CompoundLabelException ex) {
                getEditor().getKielFrame().handleException(
                    ex.getMessage(), false, ex);
            }
        }
    }
}
```

kiel.editor.controller.ActionEditStateVariables

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;
import java.util.Collection;
import java.util.StringTokenizer;

import javax.swing.JOptionPane;
import javax.swing.JTextField;

import org.jgraph.graph.DefaultGraphCell;

import kiel.dataStructure.CompositeState;
import kiel.dataStructure.StringVariable;
import kiel.dataStructure.Variable;
import kiel.dataStructure.boolexp.BooleanExpression;
import kiel.dataStructure.boolexp.BooleanVariable;
import kiel.dataStructure.doubleexp.DoubleExpression;
import kiel.dataStructure.doubleexp.DoubleVariable;
import kiel.dataStructure.floatexp.FloatExpression;
import kiel.dataStructure.floatexp.FloatVariable;
import kiel.dataStructure.intexp.IntegerExpression;
import kiel.dataStructure.intexp.IntegerVariable;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.resources.ResourceBundle;

/**
 * Shows an edit dialog to change the state's variables.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.19 $ last modified $Date: 2005/07/21 16:47:53 $
 */
public final class ActionEditStateVariables extends EditorAction {

    /**
     * Creates an object with an Edit16.gif image.
     */
    ActionEditStateVariables() {
        super("Edit16.gif");
    }

    /**
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {

        CompositeState state = (CompositeState) MyGraphConstants
            .getGraphicalObject(((DefaultGraphCell) getEditor().getGraph()
                .getSelectionCell()).getAttributes());

        String variablesText = "";
        for (int i = 0; i < state.getVariables().size(); i++) {

            Variable v = (Variable) state.getVariables().get(i);
            variablesText += v.getName();

            if (v instanceof BooleanVariable) {
                BooleanExpression expr =
                    ((BooleanVariable) v).getInitialValue();
                if (expr != null) {
                    variablesText += " := " + expr.toString();
                }
                variablesText += " : boolean";
            } else if (v instanceof DoubleVariable) {
                DoubleExpression expr =
                    ((DoubleVariable) v).getInitialValue();
                if (expr != null) {
                    variablesText += " := " + expr.toDoubleExpString();
                }
                variablesText += " : double";
            } else if (v instanceof FloatVariable) {
                FloatExpression expr =
                    ((FloatVariable) v).getInitialValue();
                if (expr != null) {
                    variablesText += " := " + expr.toFloatExpString();
                }
                variablesText += " : float";
            } else if (v instanceof IntegerVariable) {
                IntegerExpression expr =
                    ((IntegerVariable) v).getInitialValue();
                if (expr != null) {
                    variablesText += " := " + expr.toIntExpString();
                }
                variablesText += " : integer";
            } else if (v instanceof StringVariable) {
                variablesText += " : string";
            }
            if (i < state.getVariables().size() - 1) {
                variablesText += ", ";
            }
        }

        final int columns = 50;
        JTextField textfield = new JTextField(variablesText, columns);
        int answer = JOptionPane.showOptionDialog(
            getEditor().getKielFrame().getJFrame(),
            textfield,
            ResourceBundle.getString("EditStatechartVariables"),
            JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null, new Object[] {
```

```
                ResourceBundle.getString("OK"),
                ResourceBundle.getString("Cancel")},
                ResourceBundle.getString("OK"));
        if (answer == 0) {
            try {
                StringTokenizer st =
                    new StringTokenizer(textfield.getText(), ",");
                variablesText = "";
                while (st.hasMoreTokens()) {
                    String aVariable = st.nextToken();
                    if (aVariable.indexOf(":") == -1) {
                        aVariable += " : integer";
                    }
                    variablesText += aVariable;
                    if (st.hasMoreTokens()) {
                        variablesText += ", ";
                    }
                }
                getEditor().getLogFile().log(0,
                    "Parse variables: " + variablesText);
                Collection newVariables = getEditor().getKielFrame()
                    .parseVariables(variablesText);
                state.getVariables().clear();
                state.getVariables().addAll(newVariables);
                getEditor().getGraph().repaint();
            } catch (Exception ex) {
                getEditor().getKielFrame().handleException(
                    ex.getMessage(), false, ex);
            }
        }
    }
}
```

kiel.editor.controller.ActionEditTransitionPriority

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import javax.swing.JOptionPane;
import javax.swing.JTextField;

import org.jgraph.graph.DefaultGraphCell;

import kiel.dataStructure.Transition;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.graph.TransitionCell;
import kiel.editor.resources.ResourceBundle;

/**
 * Shows an edit dialog to change the state's events.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.3 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ActionEditTransitionPriority extends EditorAction {

    /**
     * Creates an object with an Edit16.gif image.
     */
    ActionEditTransitionPriority() {
        super("Edit16.gif");
    }

    /**
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {

        Transition transition = (Transition) MyGraphConstants
            .getGraphicalObject(((DefaultGraphCell) getEditor().getGraph()
                .getSelectionCell()).getAttributes());

        JTextField textField = new JTextField(
            transition.getPriority().getValue() + "", 2);
        int answer = JOptionPane.showOptionDialog(
            getEditor().getKielFrame().getJFrame(),
            textField,
            ResourceBundle.getString("editTransitionPriority"),
            JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null, new Object[] {
                ResourceBundle.getString("OK"),
                ResourceBundle.getString("Cancel")},
                ResourceBundle.getString("OK"));
        if (answer == 0) {
            try {
                getEditor().getGraph().getMyGraphModel()
                .changeTransitionPriority(
                    ((TransitionCell) getEditor().getGraph()
                    .getSelectionCell()),
                    Integer.parseInt(textField.getText()));
                getEditor().getGraph().repaint();
            } catch (NumberFormatException ex) {
                getEditor().getKielFrame().handleException(
                    ex.getMessage(), false, ex);
            }
        }
    }
}
```

kiel.editor.controller.ActionEditUndo

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;

/**
 * Implements the undo feature.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionEditUndo extends EditorAction {

    /**
     * Creates an object with an Undo16.gif image.
     */
    ActionEditUndo() {
        super("Undo16.gif");
        setEnabled(false);
    }

    /**
     * Delegates to the UndoClusterManager.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().getUndoClusterManager().undo();
    }
}
```

kiel.editor.controller.ActionFileExport

```java
package kiel.editor.controller;

import java.awt.AlphaComposite;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.Transparency;
import java.awt.color.ColorSpace;
import java.awt.event.ActionEvent;
import java.awt.geom.Rectangle2D;
import java.awt.image.BufferedImage;
import java.awt.image.ColorModel;
import java.awt.image.ComponentColorModel;
import java.awt.image.DataBuffer;
import java.awt.image.FilteredImageSource;
import java.io.ByteArrayOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;

import javax.swing.JFileChooser;
import javax.swing.RepaintManager;
import javax.swing.filechooser.FileFilter;

import kiel.editor.resources.ResourceBundle;

import org.jgraph.JGraph;
import org.jibble.epsgraphics.EpsGraphics2D;

import Acme.JPM.Encoders.GifEncoder;

import com.eteks.filter.Web216ColorsFilter;

/**
 * Provides the export formats GIF and EPS in a file chooser dialog box.
 * The code is taken from the JGraph framework.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.24 $ last modified $Date: 2005/08/01 17:53:29 $
 */
public final class ActionFileExport extends EditorAction {

    /**
     * Creates an object with an Export16.gif image.
     */
    ActionFileExport() {
        super("Export16.gif");
    }

    /**
     * Opens a file chooser dialog and provides the types GIF and EPS.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {

        final boolean isPortsVisible = getEditor().getGraph().isPortsVisible
    ();
        getEditor().getGraph().setPortsVisible(false);

        try {
            JFileChooser fileChooser = new JFileChooser(
                getEditor().getKielFrame().getFileInterfaceSaveDir());
            fileChooser.setDialogType(JFileChooser.SAVE_DIALOG);
            final String[] suffix = new String[]{".gif", ".eps"};
            final String[] description = new String[]{
                ResourceBundle.getString("GIF_-_Graphics_Interchange_Format
    "),
                ResourceBundle.getString("EPS_-_Encapsulated_Postscript")};
            FileFilter[] filters = new FileFilter[suffix.length];
            for (int i = 0; i < filters.length; i++) {
                final int ii = i;
                filters[i] = new FileFilter() {

                    public boolean accept(final File f) {

                        return f != null
                                && f.getName().toLowerCase().endsWith(
                                        suffix[ii]);
                    }

                    public String getDescription() {
                        return description[ii];
                    }
                };
                fileChooser.addChoosableFileFilter(filters[i]);
            }
            int answer = fileChooser.showDialog(getEditor()
                    .getKielFrame().getJFrame(), ResourceBundle
                    .getString("save"));
            if (answer == JFileChooser.APPROVE_OPTION) {
                File selectedFile = fileChooser.getSelectedFile();
                FileOutputStream fos = null;
                // Find the filter...
                int filterIndex = 0;
                for (int filterCount = 0;
                    filterCount < filters.length; filterCount++) {
                    if (filters[filterCount] == fileChooser
                            .getFileFilter()) {
```

```
                        filterIndex = filterCount;
                    }
                }
                if (!selectedFile.toString().toLowerCase().endsWith(
                        suffix[filterIndex])) {
                    selectedFile = new File(selectedFile
                            + suffix[filterIndex]);
                }
                fos = new FileOutputStream(selectedFile);
                if (filterIndex == 0) {
                    saveGIF(fos);
                } else if (filterIndex == 1) {
                    saveEPS(fos);
                }
                if (fos != null) {
                    // Write to file
                    fos.flush();
                    fos.close();
                }
            }
        } catch (IOException ex) {
            getEditor().getKielFrame().handleException(
                ex.getMessage(), true, ex);
        }
        getEditor().getGraph().setPortsVisible(isPortsVisible);
    }


    /**
     * Extends EpsGraphics2D in order to suppress the stack trace dump
     * at every setXORMode()-call.
     * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
     a>
     */
    static class MyEpsGraphics2D extends EpsGraphics2D {
        /**
         *
         */
        public MyEpsGraphics2D() {
            // nothing
        }

        /**
         * @param g this.
         */
        public MyEpsGraphics2D(final MyEpsGraphics2D g) {
            super(g);
        }

        /**
         * @see java.awt.Graphics#setXORMode(java.awt.Color)
         */
        public void setXORMode(final Color c1) {
//            methodNotSupported();
```

```
        }

        /**
         * @see java.awt.Graphics#create()
         */
        public Graphics create() {
            return new MyEpsGraphics2D(this);
        }
    }

    /**
     * Handles converting to EPS.
     * @param fos .
     * @throws IOException .
     */
    private void saveEPS(final FileOutputStream fos) throws IOException {

        MyEpsGraphics2D graphics = new MyEpsGraphics2D();
        graphics.setColor(getEditor().getGraph().getBackground());
        Rectangle2D bounds = getEditor().getGraph().getCellBounds(
            getEditor().getGraph().getDescendants(
                getEditor().getGraph().getRoots()));
        getEditor().getGraph().toScreen(bounds);
        Dimension d = bounds.getBounds().getSize();
        final int ten = 10;
        final int five = 5;
        graphics.fillRect(0, 0, d.width + ten, d.height + ten);
        graphics.translate(-bounds.getX() + five, -bounds.getY() + five);
        Object[] selection = getEditor().getGraph().getSelectionCells();
        boolean gridVisible = getEditor().getGraph().isGridVisible();
        boolean doubleBuffered = getEditor().getGraph()
                .isDoubleBuffered();
        getEditor().getGraph().setGridVisible(false);
        getEditor().getGraph().setDoubleBuffered(false);
        getEditor().getGraph().clearSelection();
        getEditor().getGraph().paint(graphics);
        getEditor().getGraph().setSelectionCells(selection);
        getEditor().getGraph().setGridVisible(gridVisible);
        getEditor().getGraph().setDoubleBuffered(doubleBuffered);
        fos.write(graphics.toString().getBytes());
    }

    /**
     * Handles converting to GIF.
     * @param fos .
     * @throws IOException .
     */
    private void saveGIF(final FileOutputStream fos) throws IOException {

        getEditor().getGraph().setSelectionCells(null);
        final int i = 5;
        fos.write(
            convertToGif(toImage(getEditor().getGraph(), i)));
    }
```

```java
/**
 * Taken from jgraph addons JGraphUtilities.
 * @param graph .
 * @param inset .
 * @return .
 */
public static BufferedImage toImage(final JGraph graph, final int inset)
  {
    Object[] cells = graph.getRoots();
    Rectangle2D bounds = graph.getCellBounds(cells);
    if (bounds != null) {
        graph.toScreen(bounds);
        ColorSpace cs = ColorSpace.getInstance(ColorSpace.CS_sRGB);
        ColorModel cm = new ComponentColorModel(cs, false, true,
                Transparency.OPAQUE, DataBuffer.TYPE_BYTE);
        BufferedImage img = new BufferedImage((int) bounds.getWidth() +
 2
                * inset, (int) bounds.getHeight() + 2 * inset,
                BufferedImage.TYPE_INT_ARGB);
        Graphics2D graphics = img.createGraphics();
        if (graph.isOpaque()) {
            graphics.setColor(graph.getBackground());
            graphics.fillRect(0, 0, img.getWidth(), img.getHeight());

        } else {
            graphics.setComposite(AlphaComposite.getInstance(
                    AlphaComposite.CLEAR, 0.0f));
            graphics.fillRect(0, 0, img.getWidth(), img.getHeight());
            graphics.setComposite(AlphaComposite.SrcOver);
        }
        graphics.translate((int) (-bounds.getX() + inset), (int) (-
 bounds
                .getY() + inset));
        boolean tmp = graph.isDoubleBuffered();
        RepaintManager currentManager = RepaintManager
                .currentManager(graph);
            currentManager.setDoubleBufferingEnabled(false);
            graph.paint(graphics);
            currentManager.setDoubleBufferingEnabled(true);
            return img;
        }
        return null;
    }

    /**
     * Converts Image to GIF-encoded data, reducing the number of colors if
     * needed Taken from JGraphPad.FileExportGIF.
     *
     * @param oImgBuffer .
     * @return .
     * @throws IOException .
     */
    public static byte[] convertToGif(final Image oImgBuffer)
        throws IOException {

        Graphics oGrf = oImgBuffer.getGraphics();
        ByteArrayOutputStream oOut = null;
        try {
            oOut = new ByteArrayOutputStream();
            new GifEncoder(oImgBuffer, oOut).encode();
        } catch (IOException e) {
            // GifEncoder throws IOException when GIF contains too many
            // colors
            // if this happens, filter image to reduce number of colors
            final FilteredImageSource filter = new FilteredImageSource(
                    oImgBuffer.getSource(), new Web216ColorsFilter());
            oOut = new ByteArrayOutputStream();
            new GifEncoder(filter, oOut).encode();
        }
        return oOut.toByteArray();
    }
}
```

kiel.editor.controller.ActionFileNewEsterelStudio

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.MyGraphModel;
import kiel.editor.resources.ResourceBundle;

/**
 * Creates a new data model for a Esterel Studio statechart.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.16 $ last modified $Date: 2005/08/02 18:51:48 $
 */
public final class ActionFileNewEsterelStudio extends EditorAction {

    /**
     * Creates an object with an New16.gif image.
     */
    ActionFileNewEsterelStudio() {
        super("New16.gif");
    }

    /**
     * sets a new MyGraphModel object.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().setModel(new MyGraphModel(
            ResourceBundle.getString("ActionFileNewEsterelStudio"),
            getEditor().getGraph().getGraphLayoutCache()));
        getEditor().getKielFrame().setCurrentFile(null);
    }
}
```

kiel.editor.controller.ActionFileNewMatlab

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.MyGraphModel;
import kiel.editor.resources.ResourceBundle;

/**
 * Creates a new data model for a Esterel Studio statechart.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.17 $ last modified $Date: 2005/08/02 18:52:04 $
 */
public final class ActionFileNewMatlab extends EditorAction {

    /**
     * Creates an object with an New16.gif image.
     */
    ActionFileNewMatlab() {
        super("New16.gif");
    }

    /**
     * sets a new MyGraphModel object.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().setModel(new MyGraphModel(
            ResourceBundle.getString("ActionFileNewMatlab"),
            getEditor().getGraph().getGraphLayoutCache()));
        getEditor().getKielFrame().setCurrentFile(null);
    }
}
```

kiel.editor.controller.ActionFilePreferences

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.PreferencesDialog;

/**
 * Opens the preferences dialog.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/06/22 15:19:58 $
 */
public final class ActionFilePreferences extends EditorAction {

    /**
     * Creates an object with an Preferences16.gif image.
     */
    ActionFilePreferences() {
        super("Preferences16.gif");
    }

    /**
     * Opens the preferences dialog.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        new PreferencesDialog(getEditor().getKielFrame().getJFrame())
            .setVisible(true);
    }
}
```

kiel.editor.controller.ActionFilePrint

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;
import java.awt.print.PageFormat;
import java.awt.print.PrinterException;
import java.awt.print.PrinterJob;

/**
 * Should implement the print feature, but is not implemented yet.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.16 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ActionFilePrint extends EditorAction {

    /**
     * Creates an object with an Print16.gif image.
     */
    ActionFilePrint() {
        super("Print16.gif");
    }

    /**
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        PrinterJob printJob = PrinterJob.getPrinterJob();
        printJob.setPrintable(getEditor().getGraph());
        printJob.pageDialog(new PageFormat());
        if (printJob.printDialog()) {
            try {
                printJob.print();
            } catch (PrinterException ex) {
                getEditor().getKielFrame().handleException(
                    ex.getMessage(), true, ex);
            }
        }
    }
}
```

kiel.editor.controller.ActionFileSave

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;

/**
 * Saves the current statechart to file. If no filename is associated to the
 * current statechart, then this Action delegates to ActionFileSaveAs.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.19 $ last modified $Date: 2005/06/13 21:05:15 $
 */
public final class ActionFileSave extends EditorAction {

    /**
     * Creates an object with an Save16.gif image.
     */
    ActionFileSave() {
        super("Save16.gif");
    }

    /**
     * Opens the file save dialog as provided by the KIEL-FileInterface.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        if (getEditor().getKielFrame().getCurrentFile() != null) {
            try {
                getEditor().getKielFrame().writeStateChartFile(
                    getEditor().getGraph().getMyGraphModel()
                        .getCurrentStateChart(),
                    getEditor().getGraph().getMyGraphModel()
                        .getCurrentView());
            } catch (Exception ex) {
                getEditor().getKielFrame().handleException(
                    ex.getMessage(), false, ex);
            }
        } else {
            EditorActions.get(ActionFileSaveAs.class).actionPerformed(e);
        }
    }
}
```

kiel.editor.controller.ActionFileSaveAs

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;

/**
 * Saves the current statechart to file by opening the file save dialog.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.16 $ last modified $Date: 2005/06/13 21:05:21 $
 */
public final class ActionFileSaveAs extends EditorAction {

    /**
     * Creates an object with an SaveAs16.gif image.
     */
    ActionFileSaveAs() {
        super("SaveAs16.gif");
    }

    /**
     * Saves the current statechart to file by opening the file save dialog.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {

        try {
            getEditor().getKielFrame().writeStateChartFileWithSaveDialog(
                getEditor().getGraph().getMyGraphModel().
 getCurrentStateChart(),
                getEditor().getGraph().getMyGraphModel().getCurrentView());
            if (getEditor().getKielFrame().getStatechartFile() != null) {
                getEditor().getKielFrame().setCurrentFile(
                    getEditor().getKielFrame().getStatechartFile());
            }
        } catch (Exception ex) {
            getEditor().getKielFrame().handleException(
                ex.getMessage(), false, ex);
        }
    }
}
```

kiel.editor.controller.ActionFormatAlign

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;
import java.awt.geom.Rectangle2D;
import java.util.Hashtable;

import org.jgraph.graph.AttributeMap;
import org.jgraph.graph.DefaultGraphCell;
import org.jgraph.graph.GraphConstants;

/**
 * The superclass for all align features. The subclasses set some
     coordinates
 * and then the doAction() method of this class is performed.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.18 $ last modified $Date: 2005/07/25 11:44:36 $
 */
abstract class ActionFormatAlign extends EditorAction {

    /**
     * Creates an ActionFormatAlign object.
     * @param iconName .
     */
    ActionFormatAlign(final String iconName) {
        super(iconName);
    }

    /**
     * The target bounds of the currently selected diagram elements.
     */
    private Rectangle2D bounds;

    /**
     * See bounds.
     * @return .
     */
    protected Rectangle2D getBounds() {
        return bounds;
    }

    /**
     * Called by a subclass setting a coordinate of the bounds variable.
     * @param newX .
     */
    protected void setX(final double newX) {
        bounds.setRect(newX, bounds.getY(), bounds.getWidth(),
            bounds.getHeight());
    }

/**
 * Called by a subclass setting a coordinate of the bounds variable.
 * @param newY .
 */
protected void setY(final double newY) {
    bounds.setRect(bounds.getX(), newY,
        bounds.getWidth(), bounds.getHeight());
}

/**
 * Called by a subclass setting a coordinate of the bounds variable.
 * @param newWidth .
 */
protected void setWidth(final double newWidth) {
    bounds.setRect(bounds.getX(), bounds.getY(),
        newWidth, bounds.getHeight());
}

/**
 * Called by a subclass setting a coordinate of the bounds variable.
 * @param newHeight .
 */
protected void setHeight(final double newHeight) {
    bounds.setRect(bounds.getX(), bounds.getY(),
        bounds.getWidth(), newHeight);
}

/**
 * Implements the method calls to manipulate the target bounds.
 * @param firstBounds .
 */
protected abstract void setNewCoordinates(final Rectangle2D firstBounds)
 ;

/**
 * Sets the bounds for the selected cells.
 * @see kiel.editor.controller.EditorAction#doAction(
 * java.awt.event.ActionEvent)
 */
protected void doAction(final ActionEvent e) {

    getEditor().getGraph().getUndoClusterManager()
        .setUndoClusterStart();
    Hashtable transportMap = new Hashtable();
    Object[] allSelectedCells = getEditor().getGraph()
            .getSelectionCells();
    Rectangle2D firstBounds = GraphConstants
            .getBounds(((DefaultGraphCell) allSelectedCells[0])
                .getAttributes());
    for (int i = 0; i < allSelectedCells.length; i++) {
        bounds = GraphConstants.getBounds(
```

```
        ((DefaultGraphCell) allSelectedCells[i])
                .getAttributes()).getBounds();
setNewCoordinates(firstBounds);
AttributeMap newAttributes = new AttributeMap();
GraphConstants.setBounds(
        newAttributes, new AttributeMap().createRect(bounds));
transportMap.put(allSelectedCells[i], newAttributes);
            }
        }
        getEditor().getGraph().getMyGraphModel().edit(transportMap,
                null, null, null);
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
    this);
    }
}
```

kiel.editor.controller.ActionFormatAlignBottom

```java
package kiel.editor.controller;

import java.awt.geom.Rectangle2D;

/**
 * Aligns the selected cells on the bottom border of the first selected cell
 *    .
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionFormatAlignBottom extends ActionFormatAlign {

    /**
     * Creates an object with an AlignBottom16.gif image.
     */
    ActionFormatAlignBottom() {
        super("AlignBottom16.gif");
    }

    /**
     * Set y-coordinate.
     * @see kiel.editor.controller.ActionFormatAlign
     * #setNewCoordinates(java.awt.geom.Rectangle2D)
     */
    protected void setNewCoordinates(final Rectangle2D firstBounds) {
        setY((firstBounds.getY()
            + firstBounds.getHeight()
            - getBounds().getHeight()));
    }
}
```

kiel.editor.controller.ActionFormatAlignCenterHorizontal

```java
package kiel.editor.controller;

import java.awt.geom.Rectangle2D;

/**
 * Aligns the selected cells, so that all cells' center points lie
 * on the same horizontal line.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionFormatAlignCenterHorizontal extends
     ActionFormatAlign {

    /**
     * Creates an object with an AlignCenter16.gif image.
     */
    ActionFormatAlignCenterHorizontal() {
        super("AlignCenter16.gif");
    }

    /**
     * Sets the y-coordinate.
     * @see kiel.editor.controller.ActionFormatAlign
     * #setNewCoordinates(java.awt.geom.Rectangle2D)
     */
    protected void setNewCoordinates(final Rectangle2D firstBounds) {
        setY(((firstBounds.getY()
            + firstBounds.getHeight() / 2)
            - getBounds().getHeight() / 2));
    }
}
```

kiel.editor.controller.ActionFormatAlignCenterVertical

```java
package kiel.editor.controller;

import java.awt.geom.Rectangle2D;

/**
 * Aligns the selected cells, so that all cells' center points lie
 * on the same vertical line.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionFormatAlignCenterVertical extends ActionFormatAlign
      {

    /**
     * Creates an object with an AlignCenter16.gif image.
     */
    ActionFormatAlignCenterVertical() {
        super("AlignCenter16.gif");
    }

    /**
     * Sets the x-coordinate.
     * @see kiel.editor.controller.ActionFormatAlign
     * #setNewCoordinates(java.awt.geom.Rectangle2D)
     */
    protected void setNewCoordinates(final Rectangle2D firstBounds) {
        setX(((firstBounds.getX() + firstBounds.getWidth() / 2)
            - getBounds().getWidth() / 2));
    }
}
```

kiel.editor.controller.ActionFormatAlignJustifyHorizontal

```java
package kiel.editor.controller;

import java.awt.geom.Rectangle2D;

/**
 * Sets all selected cells width to the first selected cells' width.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/08 15:49:19 $
 */
public final class ActionFormatAlignJustifyHorizontal
    extends ActionFormatAlign {

    /**
     * Creates an object with an AlignJustifyHorizontal16.gif image.
     */
    ActionFormatAlignJustifyHorizontal() {
        super("AlignJustifyHorizontal16.gif");
    }

    /**
     * Sets the width.
     * @see kiel.editor.controller.ActionFormatAlign
     * #setNewCoordinates(java.awt.geom.Rectangle2D)
     */
    protected void setNewCoordinates(final Rectangle2D firstBounds) {
        setWidth(firstBounds.getWidth());
    }
}
```

kiel.editor.controller.ActionFormatAlignJustifyVertical

```
package kiel.editor.controller;

import java.awt.geom.Rectangle2D;

/**
 * Sets all selected cells width to the first selected cells' height.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionFormatAlignJustifyVertical extends
     ActionFormatAlign {

    /**
     * Creates an object with an AlignJustifyVertical16.gif image.
     */
    ActionFormatAlignJustifyVertical() {
        super("AlignJustifyVertical16.gif");
    }

    /**
     * Sets the height.
     * @see kiel.editor.controller.ActionFormatAlign
     * #setNewCoordinates(java.awt.geom.Rectangle2D)
     */
    protected void setNewCoordinates(final Rectangle2D firstBounds) {
        setHeight(firstBounds.getHeight());
    }
}
```

kiel.editor.controller.ActionFormatAlignLeft

```
package kiel.editor.controller;

import java.awt.geom.Rectangle2D;

/**
 * Aligns the selected cells on the left border of the first selected cell.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionFormatAlignLeft extends ActionFormatAlign {

    /**
     * Creates an object with an AlignLeft16.gif image.
     */
    ActionFormatAlignLeft() {
        super("AlignLeft16.gif");
    }

    /**
     * Sets the x-coordinate.
     * @see kiel.editor.controller.ActionFormatAlign
     * #setNewCoordinates(java.awt.geom.Rectangle2D)
     */
    protected void setNewCoordinates(final Rectangle2D firstBounds) {
        setX(firstBounds.getX());
    }
}
```

kiel.editor.controller.ActionFormatAlignRight

```java
package kiel.editor.controller;

import java.awt.geom.Rectangle2D;

/**
 * Aligns the selected cells on the right border of the first selected cell.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/08 15:49:19 $
 */
public final class ActionFormatAlignRight extends ActionFormatAlign {

    /**
     * Creates an object with an AlignRight16.gif image.
     */
    ActionFormatAlignRight() {
        super("AlignRight16.gif");
    }

    /**
     * Sets the x-coordinate.
     * @see kiel.editor.controller.ActionFormatAlign
     * #setNewCoordinates(java.awt.geom.Rectangle2D)
     */
    protected void setNewCoordinates(final Rectangle2D firstBounds) {
        setX((firstBounds.getX()
            + firstBounds.getWidth()
            - getBounds().getWidth()));
    }
}
```

kiel.editor.controller.ActionFormatAlignTop

```
package kiel.editor.controller;

import java.awt.geom.Rectangle2D;

/**
 * Aligns the selected cells on the top border of the first selected cell.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionFormatAlignTop extends ActionFormatAlign {

    /**
     * Creates an object with an AlignTop16.gif image.
     */
    ActionFormatAlignTop() {
        super("AlignTop16.gif");
    }

    /**
     * Sets the y-coordinate.
     * @see kiel.editor.controller.ActionFormatAlign
     * #setNewCoordinates(java.awt.geom.Rectangle2D)
     */
    protected void setNewCoordinates(final Rectangle2D firstBounds) {
        setY(firstBounds.getY());
    }
}
```

kiel.editor.controller.ActionFormatZoom

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;

/**
 * Sets the scale factor to 1.0, means no zoom at all.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.14 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ActionFormatZoom100 extends EditorAction {

    /**
     * Creates an object with an Zoom16.gif image.
     */
    ActionFormatZoom100() {
        super("Zoom16.gif");
    }

    /**
     * Sets the zoom factor to 1.0.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().setScale(1.0);
    }
}
```

kiel.editor.controller.ActionFormatZoomIn

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.resources.Preferences;

/**
 * Sets the zoom factor by multiplying it with the preferences zoom in
 *    factor.
 * That defines steps with relative value.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.14 $ last modified $Date: 2005/03/08 15:49:19 $
 */
public final class ActionFormatZoomIn extends EditorAction {

    /**
     * Creates an object with an ZoomIn16.gif image.
     */
    ActionFormatZoomIn() {
        super("ZoomIn16.gif");
    }

    /**
     * Sets the zoom factor by multiplying it with the preferences zoom in
     * factor. That defines steps with relative value.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().setScale(Preferences.getZoomInFactor()
            * getEditor().getGraph().getScale());
    }
}
```

kiel.editor.controller.ActionFormatZoomOut

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.resources.Preferences;

/**
 * Sets the zoom factor by multiplying it with the preferences zoom out
     factor.
 * That defines steps with relative value.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.14 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionFormatZoomOut extends EditorAction {

    /**
     * Creates an object with an ZoomOut16.gif image.
     */
    ActionFormatZoomOut() {
        super("ZoomOut16.gif");
    }

    /**
     * Sets the zoom factor by multiplying it with the preferences zoom in
     * factor.
     * That defines steps with relative value.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        getEditor().getGraph().setScale(getEditor().getGraph().getScale()
            * Preferences.getZoomOutFactor());
    }
}
```

kiel.editor.controller.ActionLayout

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;
import java.util.Iterator;

import kiel.dataStructure.CompositeState;
import kiel.dataStructure.Region;
import kiel.dataStructure.StateChart;
import kiel.editor.MorphingLayouter;
import kiel.graphicalInformations.LabelLayoutInformation;
import kiel.graphicalInformations.View;
import kiel.util.main.KielLayouter;

/**
 * Layouts the current statechart with the given layouter from the layouter
 * package.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.31 $ last modified $Date: 2005/07/15 20:25:02 $
 */
public class ActionLayout extends EditorAction
    implements MorphingLayouter.MyListener {

    /**
     * The layouter name as defined by the layouter package, provided by the
     * Handler.
     */
    private KielLayouter layouter;

    /**
     * Creates an object with an layout.gif image.
     * @param aLayouterName .
     */
    public ActionLayout(final String aLayouterName) {
        super(aLayouterName, "layout.gif");
        layouter = getEditor().getKielFrame()
            .getKielLayouterByName(aLayouterName);
    }

    /**
     * Just calls layout(State).
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        layout(getEditor().getGraph().getMyGraphModel().getCurrentStateChart
     ()
            .getRootNode());
    }

/**
 * Layouts the given state with the morphing algorithm.
 * @param state .
 */
protected final void layout(final CompositeState state) {
    StateChart chart = new StateChart();
    chart.setRootNode(state);
    try {
        layouter.setStatechart(chart);
    } catch (Exception e) {
        getEditor().getKielFrame().handleException(
            e.getMessage(), false, e);
        return;
    }
    if (layouter.getStaticView() == null) {
        layouter.layoutView(View.getInitializedView(chart));
    } else {
        layouter.layoutView(layouter.getStaticView());
    }

    // getGraph().setModel(new KielGraphModel(chart,
    // layouter.getStaticView(), new
    // GraphLayoutCache(getGraph())));

    /* If the user didnt not want to layout the whole statechart (the
     * root state) then the layouter.getStaticView() does not contain
     * the layouts of all elements, so it has to be filled up.
     */
    if (layouter.getStaticView() != null) {
        // can be null if "Original layout" is chosen...
        Iterator keys = getEditor().getCurrentView().getAllKeys();
        while (keys.hasNext()) {
            String id = (String) keys.next();
            if (!layouter.getStaticView().containsId(id)) {
                layouter.getStaticView().put(id, getEditor()
                    .getCurrentView().getById(id));
            }
        }
        // the selected state's position (which has to be layouted)
        // should not change!
        layouter.getStaticView().getLayoutInformation(state)
            .setUpperLeftPoint(
                getEditor().getCurrentView().getLayoutInformation(state)
                .getUpperLeftPoint());
        if (state instanceof Region) {
            layouter.getStaticView().getLayoutInformation(state)
                .setWidth(
                    getEditor().getCurrentView().getLayoutInformation(
 state)
                    .getWidth());
            layouter.getStaticView().getLayoutInformation(state)
```

```
                .setHeight(
                    getEditor().getCurrentView().getLayoutInformation(
state)
                    .getHeight());
        }
        // Take the layouted transition label layout informations...
        keys = layouter.getStaticView().getAllKeys();
        while (keys.hasNext()) {
            String id = (String) keys.next();
            if (layouter.getStaticView().getById(id)
                instanceof LabelLayoutInformation) {

                getEditor().getCurrentView().put(
                    id, layouter.getStaticView().getById(id));
            }
        }
        // now do the layout!
        new MorphingLayouter(
            getEditor().getGraph(),
            this, true).applyNewView(layouter.getStaticView());
    }
}

/**
 * Calls UndoClusterManager.setUndoClusterEnd().
 * @see kiel.editor.MorphingLayouter.MyListener#morphingFinished()
 */
public final void morphingFinished() {
    getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
 this);
}
}
```

kiel.editor.controller.ActionLayoutSelectedState

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.dataStructure.CompositeState;
import kiel.editor.graph.CompositeStateCell;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.resources.ResourceBundle;

/**
 * Layouts the selected state with the given layouter from the layouter
 *     package.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.14 $ last modified $Date: 2005/07/14 15:27:55 $
 */
public final class ActionLayoutSelectedState extends ActionLayout {

    /**
     * Creates an object with an connecton.gif image.
     * @param aLayouterName .
     */
    public ActionLayoutSelectedState(final String aLayouterName) {
        super(aLayouterName);
    }

    /**
     * Calls layout() with the selected state.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        boolean yes = getEditor().getKielFrame().ask(
            ResourceBundle.getString("partialLayout"));

        if (yes) {
            getEditor().getCurrentStateChart(); // just for update!
            layout((CompositeState) MyGraphConstants.getGraphicalObject(
                ((CompositeStateCell) getEditor().getGraph().
 getSelectionCell())
                    .getAttributes()));
        }
    }
}
```

kiel.editor.controller.ActionViewGrid

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.resources.Preferences;

/**
 * Toggles showing the graph's grid.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.5 $ last modified $Date: 2005/03/08 15:49:19 $
 */
public final class ActionViewGrid extends EditorAction {

    /**
     * Creates an object with no image since it will be represented by a
     * checkbox.
     */
    ActionViewGrid() {
        super("grid.gif");
    }

    /**
     * Toggles the showGrid preference. All Preferences-Listeners will
     * receive a Preferences-changed message.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        Preferences.toggleShowGrid();
    }
}
```

kiel.editor.controller.ActionViewOverview

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.resources.Preferences;

/**
 * Toggles showing the overview graph.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionViewOverview extends EditorAction {

    /**
     * Creates an object with no image since it will be represented by a
     * checkbox.
     */
    ActionViewOverview() {
        super(null);
    }

    /**
     * Toggles the showOverviewLayer preference. All Preferences-Listeners
     will
     * receive a Preferences-changed message.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        Preferences.toggleShowOverviewLayer();
    }
}
```

kiel.editor.controller.ActionViewReplay

```java
package kiel.editor.controller;

import javax.swing.Action;

import kiel.editor.resources.Preferences;
import kiel.editor.resources.ResourceLoader;

/**
 * Provides the "Media-Player" feature regarding the user edit step history
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.15 $ last modified $Date: 2005/03/08 15:49:19 $
 */
abstract class ActionViewReplay extends EditorAction {

    /**
     * Is the play icon currently shown? False means the pause icon is shown
     */
    private static boolean showPlayIcon = true;

    /**
     * See showPlayIcon.
     * @return .
     */
    protected static boolean isShowPlayIcon() {
        return showPlayIcon;
    }

    /**
     * Is true if the animation is playing currently.
     */
    private static boolean isPlaying;

    /**
     * Is true if the playing direction is forward.
     */
    private static boolean forward;

    /**
     * Is true if the playing mode is fast.
     */
    private static boolean playFast;

    /**
     * Is called when the play button should flip its icon.
     */
    protected static void flipPlayPauseIcon() {
        showPlayIcon = !showPlayIcon;
        if (showPlayIcon) {
            EditorActions.get(ActionViewReplayPlay.class).putValue(
                    Action.SMALL_ICON, ResourceLoader.get("Play16.gif"));
        } else {
            EditorActions.get(ActionViewReplayPlay.class).putValue(
                    Action.SMALL_ICON, ResourceLoader.get("Pause16.gif"));
        }
    }

    /**
     * Creates an object with an connecton.gif image.
     * @param iconName .
     */
    ActionViewReplay(final String iconName) {
        super(iconName);
    }

   /**
    * Starts playing the animation.
    * @param aForward .
    * @param aPlayFast .
    */
   protected static void play(final boolean aForward, final boolean
     aPlayFast) {

       forward = aForward;
       playFast = aPlayFast;

       if (isPlaying) {
           return;
       }
       if (forward && !getEditor().getGraph().getUndoClusterManager()
               .canRedo()) {
           getEditor().getGraph().getUndoClusterManager().undoAll();
       }
       if (!forward && !getEditor().getGraph().getUndoClusterManager()
               .canUndo()) {
           getEditor().getGraph().getUndoClusterManager().redoAll();
       }
       new Thread() {
           public void run() {
               isPlaying = true;
               while (!showPlayIcon
                   &&   ((forward && getEditor().getGraph()
                           .getUndoClusterManager().canRedo()
                       || (!forward && getEditor().getGraph()
                           .getUndoClusterManager().canUndo())))) {
                   if (forward) {
                       getEditor().getGraph().getUndoClusterManager().redo()
  ;
                   } else {
                       getEditor().getGraph().getUndoClusterManager().undo()
  ;
```

```
                }

                try {
                    if (playFast) {
                        sleep(Preferences.getReplayStepTimeFast());
                    } else {
                        sleep(Preferences.getReplayStepTime());
                    }
                } catch (InterruptedException e) {
                    getEditor().getKielFrame().handleException(
                        e.getMessage(), true, e);
                }
            }
            if (!getEditor().getGraph().getUndoClusterManager().canRedo()
                || !getEditor().getGraph().getUndoClusterManager()
                    .canUndo()) {
                flipPlayPauseIcon();
            }
            isPlaying = false;
        }
    }.start();
  }
}
```

kiel.editor.controller.ActionViewReplayFastForward

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

/**
 * Starts playing in fast forward mode. See superclass for more details.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/03/08 15:49:19 $
 */
public final class ActionViewReplayFastForward extends ActionViewReplay {

    /**
     * Creates an object with an FastForward16.gif image.
     */
    ActionViewReplayFastForward() {
        super("FastForward16.gif");
    }

    /**
     * Flips the play button icon and starts playing.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        if (isShowPlayIcon()) {
            flipPlayPauseIcon();
        }
        play(true, true);
    }
}
```

kiel.editor.controller.ActionViewReplayPlay

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

/**
 * Starts playing in normal forward mode. See superclass for more details.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/03/08 15:49:19 $
 */
public final class ActionViewReplayPlay extends ActionViewReplay {

    /**
     * Creates an object with an Play16.gif image.
     */
    ActionViewReplayPlay() {
        super("Play16.gif");
    }

    /**
     * Flips the play button icon and starts playing.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        flipPlayPauseIcon();
        if (!isShowPlayIcon()) {
            play(true, false);
        }
    }
}
```

kiel.editor.controller.ActionViewReplayRewind

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

/**
 * Starts playing in normal backward mode. See superclass for more details.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionViewReplayRewind extends ActionViewReplay {

    /**
     * Creates an object with an Rewind16.gif image.
     */
    ActionViewReplayRewind() {
        super("Rewind16.gif");
    }

    /**
     * Flips the play button icon and starts playing.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        if (isShowPlayIcon()) {
            flipPlayPauseIcon();
        }
        play(false, true);
    }
}
```

kiel.editor.controller.ActionViewReplayStepBack

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

/**
 * Starts playing one step in backward mode. See superclass for more details
    .
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.15 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ActionViewReplayStepBack extends ActionViewReplay {

    /**
     * Creates an object with an StepBack16.gif image.
     */
    ActionViewReplayStepBack() {
        super("StepBack16.gif");
    }

    /**
     * Flips the play button icon and does one step backward (= one undo
     step).
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        if (!isShowPlayIcon()) {
            flipPlayPauseIcon();
        }
        if (getEditor().getGraph().getUndoClusterManager().canUndo()) {
            getEditor().getGraph().getUndoClusterManager().undo();
        }
    }
}
```

kiel.editor.controller.ActionViewReplayStepForward

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

/**
 * Starts playing one step in forward mode. See superclass for more details.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.15 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ActionViewReplayStepForward extends ActionViewReplay {

    /**
     * Creates an object with an StepForward16.gif image.
     */
    ActionViewReplayStepForward() {
        super("StepForward16.gif");
    }

    /**
     * Flips the play button icon and does one step forward (= one redo step
     ).
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        if (!isShowPlayIcon()) {
            flipPlayPauseIcon();
        }
        if (getEditor().getGraph().getUndoClusterManager().canRedo()) {
            getEditor().getGraph().getUndoClusterManager().redo();
        }
    }
}
```

kiel.editor.controller.ActionViewReplayStop

```
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.resources.Preferences;

/**
 * Stops playing the Replay. See superclass for more details.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.16 $ last modified $Date: 2005/03/08 15:49:19 $
 */
public final class ActionViewReplayStop extends ActionViewReplay {

    /**
     * Creates an object with an NavStop16.gif image.
     */
    ActionViewReplayStop() {
        super("NavStop16.gif");
    }

    /**
     * Stops the playing and does an redoAll().
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        if (!isShowPlayIcon()) {
            flipPlayPauseIcon();
            try {
                Thread.sleep(Preferences.getReplayStepTime());
            } catch (InterruptedException ex) {
                getEditor().getKielFrame().handleException(
                    ex.getMessage(), true, ex);
            }
        }
        new Thread() {
            public void run() {
                getEditor().getGraph().getUndoClusterManager().redoAll();
            }
        } .start();
    }
}
```

kiel.editor.controller.ActionViewReplayTraceLog

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.TraceLogDialog;

/**
 * Opens the trace log dialog.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.16 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ActionViewReplayTraceLog extends EditorAction {

    /**
     * Creates an object with an History16.gif image.
     */
    ActionViewReplayTraceLog() {
        super("History16.gif");
    }

    /**
     * Opens the trace log dialog.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        new TraceLogDialog(
            getEditor(),
            getEditor().getGraph().getUndoClusterManager()
                .getAllUndoableEditStepsAscending(),
            getEditor().getGraph().getUndoClusterManager()
                .getAllRedoableEditStepsAscending(),
            EditorAction.getTraceLog()).setVisible(true);
    }
}
```

kiel.editor.controller.ActionViewShowCompositeStateEvents

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.resources.Preferences;

/**
 * Toggles the preference "showCompositeStateEvents" for displaying the
 * state's variables.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.6 $ last modified $Date: 2005/03/08 15:49:19 $
 */
public final class ActionViewShowCompositeStateEvents extends EditorAction {

    /**
     * Creates an object with an group.gif image.
     */
    ActionViewShowCompositeStateEvents() {
        super("History16.gif");
    }

    /**
     * Toggles the preference "showCompositeStateEvents" for displaying the
     * state's events.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        Preferences.toggleShowCompositeStateEvents();
    }
}
```

kiel.editor.controller.ActionViewShowCompositeStateVariables

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.resources.Preferences;

/**
 * Toggles the preference "showCompositeStateVariables" for displaying the
 * state's variables.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.6 $ last modified $Date: 2005/03/08 15:49:19 $
 */
public final class ActionViewShowCompositeStateVariables extends
      EditorAction {

    /**
     * Creates an object with an group.gif image.
     */
    ActionViewShowCompositeStateVariables() {
        super("History16.gif");
    }

    /**
     * Toggles the preference "showStateActions" for displaying the
     * state's variables.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        Preferences.toggleShowCompositeStateVariables();
    }
}
```

kiel.editor.controller.ActionViewShowStateActions

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;

import kiel.editor.resources.Preferences;

/**
 * Toggles the preference "showStateActions" for displaying the
 * entry-, do- and exit-activities.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.6 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class ActionViewShowStateActions extends EditorAction {

    /**
     * Creates an object with an group.gif image.
     */
    ActionViewShowStateActions() {
        super("History16.gif");
    }

    /**
     * Toggles the preference "showStateActions" for displaying the
     * entry-, do- and exit-activities.
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected void doAction(final ActionEvent e) {
        Preferences.toggleShowStateActions();
    }
}
```

kiel.editor.controller.EditorAction

```java
package kiel.editor.controller;

import java.awt.event.ActionEvent;
import java.util.ArrayList;

import javax.swing.AbstractAction;
import javax.swing.Action;
import javax.swing.ImageIcon;

import kiel.editor.EditStep;
import kiel.editor.Editor;
import kiel.editor.resources.Preferences;
import kiel.editor.resources.ResourceBundle;
import kiel.editor.resources.ResourceLoader;

/**
 * The superclass for all user edit steps. Will add every EditorAction call
 * to the TraceLog.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.18 $ last modified $Date: 2005/08/01 17:47:28 $
 */
public abstract class EditorAction extends AbstractAction
    implements Preferences.Listener {

    /**
     * The list of EditorStep objects describing which user edit steps have
     * been done in which order.
     */
    protected static final ArrayList TRACELOG = new ArrayList();

    /**
     * @see #TRACELOG
     * @return .
     */
    public static final EditStep[] getTraceLog() {
        return (EditStep[]) TRACELOG.toArray(new EditStep[0]);
    }

    /**
     * Reference to the editor, mainly used to get access to the graph.
     * TODO Should be replaced by a reference to the graph and the kiel
     frame.
     */
    private static Editor editor;

    /**
     * Has to be called by the Editor to set the reference to itself.
     * @param anEditor .
     */
public static final void initialize(final Editor anEditor) {
    editor = anEditor;
}

/**
 * See editor.
 * @return .
 */
protected static final Editor getEditor() {
    return editor;
}

/**
 * Adds the created object to the preferences listener list.
 * @param iconName .
 */
protected EditorAction(final String iconName) {
    super(null, ResourceLoader.get(iconName));
    Preferences.addListener(this);
    preferencesChanged();
}

/**
 * Changes the Action.NAME according to the changes in the Preferences
 * class.
 * @see kiel.editor.resources.Preferences.Listener#preferencesChanged()
 */
public final void preferencesChanged() {
    putValue(Action.NAME, ResourceBundle.getString(getClass().getName()
        .substring(getClass().getName().lastIndexOf('.') + 1)));
}

/**
 * Does not add the created object to the preferences listener list.
 * @param name .
 * @param iconName .
 */
protected EditorAction(final String name, final String iconName) {
    super(name, ResourceLoader.get(iconName));
}

/**
 * The main action to be performed. This method is called just by
 * actionPerformed().
 * @param e .
 */
protected abstract void doAction(final ActionEvent e);

/**
 * The single "entry point" for starting the action.
 * @see java.awt.event.ActionListener#actionPerformed(
```

```
     * java.awt.event.ActionEvent)
     */
    public final void actionPerformed(final ActionEvent e) {
        TRACELOG.add(new EditStep(0, this, EditStep.TYPE_START));
        doAction(e);
    }

    /**
     * This will change the source of the actionevent to graph. This class
     * is taken from the examples of the JGraph framework.
     */
    final class EventRedirector extends AbstractAction {

        /**
         * The wrapped action.
         */
        private Action action;

        /**
         * Construct the "Wrapper" Action.
         * @param a .
         */
        public EventRedirector(final Action a) {
            super("", (ImageIcon) a.getValue(Action.SMALL_ICON));
            this.action = a;
        }

        /**
         * Redirect the Actionevent.
         * @see java.awt.event.ActionListener#actionPerformed(
         * java.awt.event.ActionEvent)
         */
        public void actionPerformed(final ActionEvent e) {
            action.actionPerformed(new ActionEvent(getEditor().getGraph(),
                e.getID(), e.getActionCommand(), e.getModifiers()));
        }
    }
}
```

kiel.editor.controller.EditorActions

```java
package kiel.editor.controller;

import java.util.Hashtable;

/**
 * Thats the cache for all EditorAction objects. It creates and returns
 * EditorActions. Every EditorAction is a
 * singleton, that makes it easy to enable or disable some EditorActions,
 * that means the corresponding menuItems are then disabled or enabled.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class EditorActions {

    /**
     * The cache containing all EditorActions. Mapping class --- object
     */
    private static Hashtable cache = new Hashtable();

    /**
     * Returns the EditorAction and creates it if not yet done.
     * @param actionClass The action class
     * @return The action class
     */
    public static EditorAction get(final Class actionClass) {
        EditorAction instance = (EditorAction) cache.get(actionClass);
        if (instance == null) {
            try {
                instance = (EditorAction) actionClass.newInstance();
                cache.put(actionClass, instance);
            } catch (InstantiationException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            }
        }
        return instance;
    }

    /**
     * for convenience.
     * @return .
     */
    public ActionEditCut getActionEditCut() {
        return (ActionEditCut) get(ActionEditCut.class);
    }

    /**
     * This class is not for instantiation.
     */
    private EditorActions() { }
}
```

kiel.editor.controller.EditorMode

```
package kiel.editor.controller;

import java.awt.Cursor;
import java.awt.Point;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.geom.Point2D;

import kiel.editor.resources.ResourceLoader;

/**
 * This class represents an editor state. we distinguish between the
 * Select state, the add transition state and the add node state. All these
 * states are represented as toggle buttons in the menu bar, only one per
 * time selected. This class is not just the superclass of all EditorModes,
 * it also contains some code that has to be processed in every state, that
 * is displaying the current mouse position coordinates for example. In each
 * mode we have to check if the operation is generally enabled and if an
 * intersection currently occurs. Depending on that, this class sets the
 * cursor, a red cross or a cursor representing the mode.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.19 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public abstract class EditorMode extends EditorAction {

    /**
     * Thats the red cross cursor!
     */
    private static final Cursor CURSOR_FORBIDDEN = Toolkit.getDefaultToolkit
     ()
            .createCustomCursor(ResourceLoader.get("stop.gif").getImage(),
                    new Point(0, 0), "Forbidden");

    /**
     * Thats the yellow warning cursor. This is not used yet.
    private static final Cursor CURSOR_WARNING = Toolkit.getDefaultToolkit()
            .createCustomCursor(ResourceLoader.get("attention.gif").getImage
     (),
                    new Point(0, 0), "Warning");
     */

    /**
     * This is true, if isValidPosition() returns true and if the requested
     * operation is valid, means: moving a transition for example is
     * disallowed!
     */
    private boolean isModeEnabled = true;

    /**
     * Does a node currently intersects another node?
     */
    private boolean isIntersects = false;

    /**
     * Holds a reference to the EditorMode dependent cursor.
     */
    private Cursor myCursor = Cursor.getDefaultCursor();

    /**
     * See myCursor.
     * @return .
     */
    protected final Cursor getMyCursor() {
        return myCursor;
    }

    /**
     * See myCursor.
     * @param newMyCursor .
     */
    protected final void setMyCursor(final Cursor newMyCursor) {
        myCursor = newMyCursor;
    }

    /**
     * See isIntersects.
     * @return .
     */
    protected final boolean isIntersects() {
        return isIntersects;
    }

    /**
     * Creates an EditorMode.
     * @param iconName .
     */
    protected EditorMode(final String iconName) {
        super(iconName);
    }

    /**
     * Calls a done() on the last EditorMode and sets this object as
     * current EditorMode (EditorMode.setCurrent()).
     *
     */
    public void start() {
        if (EditorModes.getCurrent() != null) {
            EditorModes.getCurrent().done();
        }
        EditorModes.setCurrent(this);
```

```java
        if (EditorModes.getCurrent() == EditorModes.get(
                EditorModeSelectOrDragOrResizeOrFlipExpand.class)) {
            getEditor().setSelectionButtonSelected();
        }
    }

    /**
     * Is called when this EditorMode is replaced by the "next" EditorMode.
     */
    public abstract void done();

    /**
     * Sets the current cursor as well.
     * @param b .
     */
    protected final void setIntersects(final boolean b) {
        isIntersects = b;
        setCursorEnabled(isModeEnabled() && !isIntersects());
    }

    /**
     * Sets the current cursor as well.
     * @param b .
     */
    protected final void setModeEnabled(final boolean b) {
        isModeEnabled = b;
        setCursorEnabled(isModeEnabled() && !isIntersects());
    }

    /**
     * Sets the red cross cursor or the EditorMode dependent cursor.
     * @param b .
     */
    private void setCursorEnabled(final boolean b) {
        if (b) {
            getEditor().getGraph().setCursor(myCursor);
        } else {
            getEditor().getGraph().setCursor(CURSOR_FORBIDDEN);
        }
    }

    /**
     * See isModeEnabled.
     * @return .
     */
    protected final boolean isModeEnabled() {
        return isModeEnabled;
    }

    /**
     * Just calls start().
     * @see kiel.editor.controller.EditorAction#doAction(
     * java.awt.event.ActionEvent)
     */
    protected final void doAction(final ActionEvent e) {
        start();
    }

    /**
     * Sets the cursor position coordinates.
     * @param p .
     * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
     a>
     */
    public void mouseMoved(final Point2D p) {
        getEditor().setLabelCursorPosition(p);
    }

    /**
     * Sets the cursor position coordinates.
     * @param p .
     */
    public void mouseDragged(final Point2D p) {
        getEditor().setLabelCursorPosition(p);
    }

    /**
     * Does nothing.
     * @param p .
     * @param isControlDown .
     */
    public void mousePressed(final Point2D p, final boolean isControlDown) {
    }

    /**
     * Does nothing.
     * @param p .
     */
    public void mouseReleased(final Point2D p) {
    }
}
```

kiel.editor.controller.EditorModeAddDeepHistory

```
package kiel.editor.controller;

import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;

import org.jgraph.graph.DefaultGraphCell;
import org.jgraph.graph.GraphCell;
import org.jgraph.graph.GraphConstants;

import kiel.dataStructure.DeepHistory;
import kiel.editor.graph.DeepHistoryCell;
import kiel.editor.graph.ORStateCell;
import kiel.editor.graph.RegionCell;
import kiel.editor.resources.Preferences;
import kiel.graphicalInformations.Properties;

/**
 * The state in which the user can add a DeepHistory.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.21 $ last modified $Date: 2005/07/27 09:42:14 $
 */
public final class EditorModeAddDeepHistory extends EditorModeAddNode {

    /**
     * Creates an object with an DeepHistory.gif image.
     */
    EditorModeAddDeepHistory() {
        super("DeepHistory.gif", DeepHistory.class);
    }

    /**
     * Returns true if the selected state is an ANDState or ORState.
     * @see kiel.editor.controller.EditorModeAddNode#isValidPosition(
     * java.awt.geom.Point2D)
     */
    protected boolean isValidPosition(final Point2D p) {
        // is allowed only once in each ANDState and ORState
        return getEditor().getGraph().getMyGraphModel().canBeAdded(
                DeepHistoryCell.class, p,
                new Class[]{RegionCell.class, ORStateCell.class}, true);
    }

    /**
     * Calls MyGraphModel.add().
     * @see kiel.editor.controller.EditorModeAddNode#doPlaceAction(
     * java.awt.geom.Point2D)
     */
    protected void doPlaceAction(final Point2D p) {

        // has to be placed in the upper left corner
        Object cell = getEditor().getGraph().getMyGraphModel()
            .getSmallestCellOfType(GraphCell.class,
                new Rectangle2D.Double(p.getX(), p.getY(), 1, 1), false);

        getEditor().getGraph().getMyGraphModel().addNode(
            getNodeClass(),
            GraphConstants.getBounds(
                ((DefaultGraphCell) cell).getAttributes()).getX()
                + Preferences.getPlaceOfDeepHistory().x
                + Preferences.getIntersectionSensitivity(),
                GraphConstants.getBounds(
                    ((DefaultGraphCell) cell).getAttributes()).getY()
                + Preferences.getPlaceOfDeepHistory().y
                + Properties.getCompositeStateHeaderHeight(),
                getEditor().getGraph().getGraphLayoutCache());
    }

    /**
     * @see kiel.editor.controller.EditorModeAddNode#useNodeEnlarger()
     */
    protected boolean useNodeEnlarger() {
        return false;
    }
}
```

kiel.editor.controller.EditorModeAddDelimiter

```java
package kiel.editor.controller;

import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;

import kiel.editor.MyGraphModelUtilities;
import kiel.editor.graph.CompositeStateCell;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.graph.NodeCell;

import org.jgraph.graph.AttributeMap;
import org.jgraph.graph.GraphCell;
import org.jgraph.graph.GraphConstants;

/**
 * The state in which the user can add a Delimiter.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.25 $ last modified $Date: 2005/07/27 09:42:14 $
 */
public abstract class EditorModeAddDelimiter extends EditorModeAddNode {

    /**
     * Creates an EditorMode object.
     * @param cursorFilename .
     * @param anIsHorizontal .
     */
    EditorModeAddDelimiter(final String cursorFilename,
        final boolean anIsHorizontal) {

        super(cursorFilename, null);
        isHorizontal = anIsHorizontal;
    }

    /**
     * Depending on the subclass. We could have used here polymorphism as
     well.
     */
    private boolean isHorizontal;

    /**
     * Returns true if the selected state is not the root state or if the.
     * cursor is not in the valid area of the selected state
     * @see kiel.editor.controller.EditorModeAddNode#isValidPosition(
     * java.awt.geom.Point2D)
     */
    protected final boolean isValidPosition(final Point2D p) {

        CompositeStateCell cell = (CompositeStateCell) getEditor().getGraph
      ()
            .getMyGraphModel().getSmallestCellOfType(
                CompositeStateCell.class,
                new AttributeMap.SerializableRectangle2D(
                    p.getX(), p.getY(), 1, 1), true);
        if (cell == null
              || getEditor().getGraph().getMyGraphModel().getRootCell()
                        == cell
                || MyGraphConstants.isCollapsed(cell.getAttributes())
                || !MyGraphModelUtilities.getValidArea(cell).contains(p)) {
            return false;
        } else {
            GraphCell[] children = (GraphCell[]) ((CompositeStateCell) cell)
                .getChildren().toArray(new GraphCell[0]);
            for (int i = 0; i < children.length; i++) {
                if (children[i] instanceof NodeCell) {
                    Rectangle2D bounds = GraphConstants.getBounds(
                            children[i].getAttributes()).getBounds();
                    if ((isHorizontal
                            && bounds.getY() < p.getY()
                            && bounds.getY()
                                + bounds.getHeight() > p.getY())
                        || (!isHorizontal
                            && bounds.getX() < p.getX()
                            && bounds.getX()
                                + bounds.getWidth() > p.getX())) {
                        return false;
                    }
                }
            }
        }
        return true;
    }

    /**
     * Calls MyGraphModel.addDelimiter().
     * @see kiel.editor.controller.EditorModeAddNode#doPlaceAction(
     * java.awt.geom.Point2D)
     */
    protected final void doPlaceAction(final Point2D p) {

        if (isHorizontal) {
            getEditor().getGraph().getMyGraphModel().addDelimiter(p,
                    new Point2D.Double(p.getX() + 1, p.getY()));
        } else {
            getEditor().getGraph().getMyGraphModel().addDelimiter(p,
                    new Point2D.Double(p.getX(), p.getY() + 1));
        }
    }

    /**
     * @see kiel.editor.controller.EditorModeAddNode#useNodeEnlarger()
```

```
 */
protected final boolean useNodeEnlarger() {
    return false;
                                                                }
                                                                  }
```

kiel.editor.controller.EditorModeAddDynamicChoice

```
package kiel.editor.controller;

import kiel.dataStructure.DynamicChoice;

/**
 * The state in which the user can add a DeepHistory.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/03/08 15:49:19 $
 */
public final class EditorModeAddDynamicChoice extends EditorModeAddNode {

    /**
     * Creates an object with an DynamicChoice.gif image.
     */
    EditorModeAddDynamicChoice() {
        super("DynamicChoice.gif", DynamicChoice.class);
    }

    /**
     * @see kiel.editor.controller.EditorModeAddNode#useNodeEnlarger()
     */
    protected boolean useNodeEnlarger() {
        return true;
    }
}
```

kiel.editor.controller.EditorModeAddFinalSimpleState

```java
package kiel.editor.controller;

import kiel.dataStructure.FinalSimpleState;

/**
 * The state in which the user can add a DeepHistory.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class EditorModeAddFinalSimpleState extends EditorModeAddNode {

    /**
     * Creates an object with an FinalSimpleState.gif image.
     */
    EditorModeAddFinalSimpleState() {
        super("FinalSimpleState.gif", FinalSimpleState.class);
    }

    /**
     * @see kiel.editor.controller.EditorModeAddNode#useNodeEnlarger()
     */
    protected boolean useNodeEnlarger() {
        return true;
    }
}
```

kiel.editor.controller.EditorModeAddHistory

```
package kiel.editor.controller;

import kiel.dataStructure.History;

/**
 * The state in which the user can add a DeepHistory.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class EditorModeAddHistory extends EditorModeAddNode {

    /**
     * Creates an object with an History.gif image.
     */
    EditorModeAddHistory() {
        super("History.gif", History.class);
    }

    /**
     * @see kiel.editor.controller.EditorModeAddNode#useNodeEnlarger()
     */
    protected boolean useNodeEnlarger() {
        return true;
    }
}
```

kiel.editor.controller.EditorModeAddHorizontalDelimiter

```
package kiel.editor.controller;

/**
 * The state in which the user can add a horizontal delimiter line.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.14 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class EditorModeAddHorizontalDelimiter
        extends EditorModeAddDelimiter {

    /**
     * Creates an object with an HorizontalDelimiter.gif image.
     */
    EditorModeAddHorizontalDelimiter() {
        super("HorizontalDelimiter.gif", true);
    }
}
```

kiel.editor.controller.EditorModeAddInitialState

```java
package kiel.editor.controller;

import java.awt.geom.Point2D;

import kiel.dataStructure.InitialState;
import kiel.editor.graph.InitialStateCell;
import kiel.editor.graph.ORStateCell;
import kiel.editor.graph.RegionCell;

/**
 * The state in which the user can add a DeepHistory.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.14 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class EditorModeAddInitialState extends EditorModeAddNode {

    /**
     * Creates an object with an InitialState.gif image.
     */
    EditorModeAddInitialState() {
        super("InitialState.gif", InitialState.class);
    }

    /**
     * Returns true if the selected node is a Region or ORState.
     * @see kiel.editor.controller.EditorModeAddNode#isValidPosition(
     * java.awt.geom.Point2D)
     */
    protected boolean isValidPosition(final Point2D p) {

        // at most one time in each ORState and Region
        return getEditor().getGraph().getMyGraphModel().canBeAdded(
                InitialStateCell.class, p,
                new Class[]{RegionCell.class, ORStateCell.class}, false);
    }

    /**
     * @see kiel.editor.controller.EditorModeAddNode#useNodeEnlarger()
     */
    protected boolean useNodeEnlarger() {
        return true;
    }
}
```

kiel.editor.controller.EditorModeAddNode

```java
package kiel.editor.controller;

import java.awt.Point;
import java.awt.Rectangle;
import java.awt.Toolkit;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;

import javax.swing.Action;
import javax.swing.ImageIcon;

import kiel.dataStructure.State;
import kiel.editor.EditStep;
import kiel.editor.MorphingLayouter;
import kiel.editor.MyGraphModelUtilities;
import kiel.editor.graph.CompositeStateCell;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.graph.NodeCell;
import kiel.editor.graph.ORStateCell;
import kiel.editor.graph.RegionCell;
import kiel.editor.resources.Preferences;
import kiel.editor.resources.ResourceLoader;
import kiel.graphicalInformations.Properties;

import org.jgraph.graph.AttributeMap;
import org.jgraph.graph.DefaultGraphCell;
import org.jgraph.graph.GraphConstants;

/**
 * The state in which the user can add a node.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.40 $ last modified $Date: 2005/07/27 09:42:14 $
 */
public abstract class EditorModeAddNode extends EditorMode {

    /**
     * The handler for the editor layout mechanism for "creating editor
     space".
     * This class does not contain the algorithm but the call to the
     * MorphingLayouter which contains that layouting information
     */
    private final NodeEnlarger nodeEnlarger = new NodeEnlarger();

    /**
     * Each EditorModeAddNode object will create a node of a particular type
     .
     */
    private final Class nodeClass;

    /**
     * See nodeClass.
     * @return .
     */
    protected final Class getNodeClass() {
        return nodeClass;
    }

    /**
     * Creates an object with the appropriate cursor.
     * @param cursorFilename .
     * @param aNodeClass .
     */
    protected EditorModeAddNode(final String cursorFilename,
        final Class aNodeClass) {

        super(cursorFilename);
        this.nodeClass = aNodeClass;

        if (cursorFilename != null) {

            ImageIcon icon = ResourceLoader.get(
                cursorFilename.substring(0, cursorFilename.lastIndexOf("."))
                    + "AndCursor.gif");

            if (icon != null) {
                setMyCursor(Toolkit.getDefaultToolkit()
                        .createCustomCursor(icon.getImage(),
                                new Point(0, 0),
                                (String) getValue(Action.NAME)));
            } else {
                setMyCursor(Toolkit.getDefaultToolkit()
                        .createCustomCursor(
                                ResourceLoader.get(cursorFilename).getImage
 (),
                                new Point(0, 0),
                                (String) getValue(Action.NAME)));
            }
        }
    }

    /**
     * Sets the appropriate cursor and delegates this call to the node
     enlarger.
     * @see kiel.editor.controller.EditorMode#start()
     */
    public final void start() {
        super.start();
        getEditor().getGraph().setCursor(getMyCursor());
        getEditor().getGraph().setPortsVisible(true);
        EditorActions.get(ActionAddTransitionToggle.class).doAction(null);
```

```java
        EditorActions.get(ActionAddTransitionToggle.class).setEnabled(false)
      ;
        nodeEnlarger.start();
    }

    /**
     * Delegates to the node enlarger.
     * @see kiel.editor.controller.EditorMode#done()
     */
    public final void done() {
        EditorActions.get(ActionAddTransitionToggle.class).doAction(null);
        EditorActions.get(ActionAddTransitionToggle.class).setEnabled(true);
        nodeEnlarger.done();
    }

    /**
     * Returns true if the currently selected state is a CompositeState.
     * @param p .
     * @return .
     */
    protected boolean isValidPosition(final Point2D p) {

        // no limit for CompositeStates
        Point2D snapPoint = getEditor().getGraph().snap(p);
        Object cell = getEditor().getGraph().getMyGraphModel()
            .getSmallestCellOfType(NodeCell.class,
                new AttributeMap.SerializableRectangle2D(
                    snapPoint.getX(), snapPoint.getY(), 1, 1), true);
        return cell instanceof CompositeStateCell
            && !MyGraphConstants.isCollapsed(
                ((DefaultGraphCell) cell).getAttributes());
    }

    /**
     * Just calls MyGraphModel.add().
     * @param p .
     */
    protected void doPlaceAction(final Point2D p) {
        Point2D snapPoint = getEditor().getGraph().snap(p);
        getEditor().getGraph().getMyGraphModel().addNode(
            nodeClass, snapPoint.getX(), snapPoint.getY(),
            getEditor().getGraph().getGraphLayoutCache());
    }

    /**
     * Calls the doPlaceAction() if isModeEnabled and not isIntersects.
     * Switches then to the Select-EditorMode.
     * @see kiel.editor.controller.EditorMode
     * #mouseReleased(java.awt.geom.Point2D)
     */
    public final void mouseReleased(final Point2D p) {
        super.mouseReleased(p);
        if (isModeEnabled() && !isIntersects()) {
            doPlaceAction(p);
                actionSuccessfulDone = true;
            }
            mouseIsAlreadyReleased = true;
            terminateEditCluster();
            EditorModes.get(EditorModeSelectOrDragOrResizeOrFlipExpand.class)
            .start();
    }

    /**
     * Reference to the currently selected state.
     */
    private static CompositeStateCell currentlySelectedCompositeState;

    /**
     * See currentlySelectedCompositeState.
     * @return .
     */
    protected static CompositeStateCell getCurrentlySelectedCompositeState()
      {
        return currentlySelectedCompositeState;
    }

    /**
     * how many MorphingLayouter calls have not been finished yet? The
     current
     * edit step cluster can be finished only if there are none left.
     */
    private int pendingMorphingRequestsCount;

    /**
     * Has mouseReleased() already been called? The current
     * edit step cluster can be finished only if there are none left.
     */
    private boolean mouseIsAlreadyReleased;

    /**
     * If the action has been done successfully, that means, mouse released
     * when the red cross was not visible, then the edit cluster can be
     * terminated, otherwise it will be discarded.
     */
    private boolean actionSuccessfulDone;

    /**
     * Sets the currentlySelectedCompositeState and starts the undo cluster.
     * Then delegates the call to the Node enlarger.
     * @see kiel.editor.controller.EditorMode
     * #mousePressed(java.awt.geom.Point2D, boolean)
     */
    public final void mousePressed(final Point2D p,
        final boolean isControlDown) {

        super.mousePressed(p, isControlDown);
        currentlySelectedCompositeState = (CompositeStateCell) getEditor()
            .getGraph().getMyGraphModel().getSmallestCellOfType(
```

```
                CompositeStateCell.class,
                new AttributeMap.SerializableRectangle2D(
                    p.getX(), p.getY(), 1, 1), false);

        getEditor().getGraph().getUndoClusterManager().setUndoClusterStart()
    ;
        pendingMorphingRequestsCount = 0;
        mouseIsAlreadyReleased = false;
        actionSuccessfulDone = false;

        nodeEnlarger.mousePressed(p);
    }

    /**
     * Sets the cursor and delegates the call to the NodeEnlarger.
     * @see kiel.editor.controller.EditorMode
     * #mouseDragged(java.awt.geom.Point2D)
     */
    public final void mouseDragged(final Point2D p) {
        super.mouseDragged(p);
        // Check if allowed to place
        setModeEnabled(isValidPosition(p));
        setIntersects(getEditor().getGraph().getMyGraphModel().intersect(
            nodeClass,
                p.getX(), p.getY()));
        nodeEnlarger.mouseDragged(p);
    }

    /**
     * Just sets the cursor.
     * @see kiel.editor.controller.EditorMode
     * #mouseMoved(java.awt.geom.Point2D)
     */
    public final void mouseMoved(final Point2D p) {
        super.mouseMoved(p);
        // Check if allowed to place
        setModeEnabled(isValidPosition(p));
        setIntersects(
            !willEnlargeTargetParent(getNewNodeBounds(p))
            && getEditor().getGraph().getMyGraphModel().intersect(
            nodeClass, p.getX(), p.getY()));
    }

    /**
     * .
     * @return .
     */
    protected abstract boolean useNodeEnlarger();

    /**
     *
     * @param newNodeBounds .
     * @return .
     */
    private boolean willEnlargeTargetParent(
        final Rectangle2D newNodeBounds) {

        DefaultGraphCell currentNode =
            (DefaultGraphCell) getEditor().getGraph().getMyGraphModel()
                .getSmallestCellOfType(CompositeStateCell.class,
                    new Rectangle(
                        (int) newNodeBounds.getX(),
                        (int) newNodeBounds.getY(), 1, 1), false);
        return
            Preferences.getLayoutOnInsert()
            && currentNode instanceof ORStateCell
            && getEditor().getGraph().getMyGraphModel().intersect(
                currentNode, newNodeBounds, false);
    }

    /**
     *
     * @param upperLeft .
     * @return .
     */
    private Rectangle2D getNewNodeBounds(final Point2D upperLeft) {
        final int sensitivityFactor = 2;
        return new AttributeMap.SerializableRectangle2D(
            upperLeft.getX() - Preferences.getIntersectionSensitivity(),
            upperLeft.getY() - Preferences.getIntersectionSensitivity(),
// TODO Why 2 times Preferences.getGridSize()??
            MyGraphModelUtilities.getInitialDimensionOf(nodeClass)
                    .width
                + sensitivityFactor * Preferences.getIntersectionSensitivity
    ()
                + sensitivityFactor * Preferences.getGridSize(),
                MyGraphModelUtilities.getInitialDimensionOf(nodeClass)
                    .height
                + sensitivityFactor * Preferences.getIntersectionSensitivity
    ()
                + sensitivityFactor * Preferences.getGridSize());
    }

    /**
     * Handles "create space!" requests while choosing a place to create
     * a new node. This class does not implement any layout algorithms but
     just
     * calls the MorphingLayouter who implements those layout mechanisms.
     * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
     a>
     */
    private final class NodeEnlarger implements MorphingLayouter.MyListener
     {

        /**
         * The initial bounds of the currently selected CompositeState
         * have to be stored.
         */
```

```java
    private Rectangle2D initialBoundsOfCurrentlySelectedCompositeState;

    /**
     * Sets the initial bounds of the currently selected CompositeState
     * and delegates the call to the NodeEnlarger.
     * @see kiel.editor.controller.EditorMode
     * #mousePressed(java.awt.geom.Point2D)
     */
    public void mousePressed(final Point2D p) {
        if (EditorModeAddNode.getCurrentlySelectedCompositeState()
                != null) {
            initialBoundsOfCurrentlySelectedCompositeState =
                GraphConstants.getBounds(
                    EditorModeAddNode.getCurrentlySelectedCompositeState
()
                    .getAttributes()).getBounds2D();
        } else {
            initialBoundsOfCurrentlySelectedCompositeState = null;
        }
        if (!(EditorModeAddNode.getCurrentlySelectedCompositeState()
            instanceof RegionCell)) {
            enlargeTargetParent(p, true);
        }
    }

    /**
     * Just delegates the call to the NodeEnlarger.
     * @see kiel.editor.controller.EditorMode
     * #mouseDragged(java.awt.geom.Point2D)
     */
    public void mouseDragged(final Point2D p) {
        enlargeTargetParent(p, false);
    }

    /**
     * Just sets the JGraph.moveable attribute to false. Don't really
     * know why.
     * @see kiel.editor.controller.EditorMode#start()
     */
    public void start() {
        getEditor().getGraph().setMoveable(false);
    }

    /**
     * Just sets the JGraph.moveable attribute to true. Don't really
     * know why.
     * @see kiel.editor.controller.EditorMode#done()
     */
    public void done() {
        getEditor().getGraph().setMoveable(true);
    }

    /**
     * Calls the MorphingLayouter depending on the Preferences.
     * @param p .
     * @param doMorphing .
     */
    private void enlargeTargetParent(final Point2D p,
        final boolean doMorphing) {

        if (!Preferences.getLayoutOnInsert()
            || !isModeEnabled()
            || !useNodeEnlarger()) {
            return;
        }
        //@TODO do the same for dragging after creation

        Rectangle2D newStateBounds = getNewNodeBounds(p);

        if (willEnlargeTargetParent(newStateBounds)) {

            //@TODO Calculate needed bounds for the current parent and
            // morph...
            Rectangle2D targetParentBounds =
                initialBoundsOfCurrentlySelectedCompositeState.getBounds
();

            // think of the or state header...
            newStateBounds.add(new AttributeMap.SerializablePoint2D(
                newStateBounds.getX(),
                newStateBounds.getY()
                - Properties.getUpperOffset(
                    (State) MyGraphConstants.getGraphicalObject(
                        EditorModeAddNode
                        .getCurrentlySelectedCompositeState()
                            .getAttributes()))));
            //...

            targetParentBounds.add(newStateBounds);

            pendingMorphingRequestsCount++;
            new MorphingLayouter(getEditor().getGraph(), this,
doMorphing)
                .expandCompositeState((DefaultGraphCell)
                    EditorModeAddNode.getCurrentlySelectedCompositeState
(),
                    targetParentBounds);

            EditorModes.getCurrent().setIntersects(false);
        }
    }

    /**
     * Decreases the pendingMorphingRequestsCount and calls
     * terminateEditCluster().
     * @see kiel.editor.MorphingLayouter.MyListener#morphingFinished()
     */
    public void morphingFinished() {
```

```
            pendingMorphingRequestsCount--;
            terminateEditCluster();
        }
    }

    /**
     * If the pendingMorphingRequestsCount is 0 and the mouse is already
     * released, then if the action was successgfully done, then terminate
     the
     * the current edit step cluster otherwise discard the current edit step
     * cluster.
     */
    private void terminateEditCluster() {
        if (pendingMorphingRequestsCount == 0 && mouseIsAlreadyReleased) {
            if (actionSuccessfulDone) {
                getEditor().getGraph().getUndoClusterManager()
                .setUndoClusterEnd(EditorModeAddNode.this);
                EditorAction.TRACELOG.add(
                    new EditStep(0, this, EditStep.TYPE_SUCCESSFUL));
            } else {
                getEditor().getGraph().getUndoClusterManager()
                .discardCurrentCluster();
                EditorAction.TRACELOG.add(
                    new EditStep(0, this, EditStep.TYPE_CANCELLED));
            }
        }
    }
}
```

kiel.editor.controller.EditorModeAddOrState

```java
package kiel.editor.controller;

import kiel.dataStructure.ORState;

/**
 * The state in which the user can add an or state.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/08 15:49:19 $
 */
public final class EditorModeAddOrState extends EditorModeAddNode {

    /**
     * Creates an object with an OrState.gif image.
     */
    EditorModeAddOrState() {
        super("OrState.gif", ORState.class);
    }

    /**
     * @see kiel.editor.controller.EditorModeAddNode#useNodeEnlarger()
     */
    protected boolean useNodeEnlarger() {
        return true;
    }
}
```

kiel.editor.controller.EditorModeAddSimpleState

```
package kiel.editor.controller;

import kiel.dataStructure.SimpleState;

/**
 * The state in which the user can add a simple state.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class EditorModeAddSimpleState extends EditorModeAddNode {

    /**
     * Creates an object with an SimpleState.gif image.
     */
    EditorModeAddSimpleState() {
        super("SimpleState.gif", SimpleState.class);
    }

    /**
     * @see kiel.editor.controller.EditorModeAddNode#useNodeEnlarger()
     */
    protected boolean useNodeEnlarger() {
        return true;
    }
}
```

kiel.editor.controller.EditorModeAddSuspend

```java
package kiel.editor.controller;

import kiel.dataStructure.Suspend;

/**
 * The state in which the user can add a suspend.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/08 15:49:19 $
 */
public final class EditorModeAddSuspend extends EditorModeAddNode {

    /**
     * Creates an object with an Suspend.gif image.
     */
    EditorModeAddSuspend() {
        super("Suspend.gif", Suspend.class);
    }

    /**
     * @see kiel.editor.controller.EditorModeAddNode#useNodeEnlarger()
     */
    protected boolean useNodeEnlarger() {
        return true;
    }
}
```

kiel.editor.controller.EditorModeAddTransition

```
package kiel.editor.controller;

import java.awt.geom.Point2D;

/**
 * The state in which the user can add a transition.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.14 $ last modified $Date: 2005/03/08 15:49:18 $
 */
public final class EditorModeAddTransition extends EditorMode {

    /**
     * Creates an object with no image since it will have no representation
     at
     * all.
     */
    EditorModeAddTransition() {
        super(null);
    }

    /**
     *
     * @see kiel.editor.controller.EditorMode#start()
     */
    public void start() {
        super.start();
        getEditor().getGraph().getUndoClusterManager().setUndoClusterStart()
      ;
    }

    /**
     * @see kiel.editor.controller.EditorModeAddNode#doPlaceAction(
     * java.awt.geom.Point2D)
     */
    protected void doPlaceAction(final Point2D p) {
    }

    /**
     *
     * @see kiel.editor.controller.EditorMode#mouseReleased(
     * java.awt.geom.Point2D)
     */
    public void mouseReleased(final Point2D p) {
        super.mouseReleased(p);
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
     this);
        EditorModes.get(EditorModeSelectOrDragOrResizeOrFlipExpand.class)
        .start();
    }

    /**
     *
     * @see kiel.editor.controller.EditorMode#done()
     */
    public void done() {
    }
}
```

kiel.editor.controller.EditorModeAddVerticalDelimiter

```java
package kiel.editor.controller;

/**
 * The state in which the user can add a vertical delimiter.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/08 15:49:19 $
 */
public final class EditorModeAddVerticalDelimiter
        extends EditorModeAddDelimiter {

    /**
     * Creates an object with an VerticalDelimiter.gif image.
     */
    EditorModeAddVerticalDelimiter() {
        super("VerticalDelimiter.gif", false);
    }
}
```

kiel.editor.controller.EditorModes

```
package kiel.editor.controller;

import java.util.Hashtable;

/**
 * The editor modes cache.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class EditorModes {

    /**
     * Comment for <code>cache</code>.
     */
    private static Hashtable cache = new Hashtable();

    /**
     * @param actionClass The action class
     * @return The action class
     */
    public static EditorMode get(final Class actionClass) {

        EditorMode instance = (EditorMode) cache.get(actionClass);
        if (instance == null) {
            try {
                instance = (EditorMode) actionClass.newInstance();
                cache.put(actionClass, instance);
            } catch (InstantiationException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            }
        }
        return instance;
    }

    /**
     * .
     */
    private EditorModes() {

    }

    /**
     * Comment for <code>currentMode</code>.
     */
    private static EditorMode currentMode =
        EditorModes.get(EditorModeSelectOrDragOrResizeOrFlipExpand.class);

    /**
     * @return .
     */
    public static EditorMode getCurrent() {
        return currentMode;
    }

    /**
     * @param anEditorMode .
     */
    static void setCurrent(final EditorMode anEditorMode) {
        currentMode = anEditorMode;
    }
}
```

kiel.editor.controller.EditorModeSelectOrDragOrResizeOrFlipExpand

```java
package kiel.editor.controller;

import java.awt.Rectangle;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.util.Arrays;
import java.util.Iterator;

import kiel.dataStructure.ANDState;
import kiel.dataStructure.GraphicalObject;
import kiel.dataStructure.ORState;
import kiel.editor.MorphingLayouter;
import kiel.editor.MyGraphModel;
import kiel.editor.MyGraphModelUtilities;
import kiel.editor.graph.ANDStateCell;
import kiel.editor.graph.CompositeStateCell;
import kiel.editor.graph.DeepHistoryCell;
import kiel.editor.graph.DelimiterLineCell;
import kiel.editor.graph.MyGraphConstants;
import kiel.editor.graph.NodeCell;
import kiel.editor.graph.NodeView;
import kiel.editor.graph.ORStateCell;
import kiel.editor.graph.Previewable;
import kiel.editor.graph.PseudoStateCell;
import kiel.editor.graph.RegionCell;
import kiel.editor.graph.TransitionCell;
import kiel.editor.resources.Preferences;
import kiel.graphicalInformations.Properties;

import org.jgraph.graph.AttributeMap;
import org.jgraph.graph.CellView;
import org.jgraph.graph.DefaultGraphCell;
import org.jgraph.graph.DefaultGraphModel;
import org.jgraph.graph.GraphConstants;

/**
 * The state in which the user can select or drag or resize or flip expand
 * a state.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.60 $ last modified $Date: 2005/08/02 19:28:13 $
 */
public class EditorModeSelectOrDragOrResizeOrFlipExpand extends EditorMode
    implements MorphingLayouter.MyListener {

    /**
     * Comment for <code>draggedCell</code>.
     */
    private Object[] selectedCells;

    /**
     *
     */
    private Rectangle2D[] selectedBounds;

    /**
     *
     */
    private boolean automaticSelectionEnabled = true;

    /**
     *
     */
    private boolean isDraggingOrResizing;

    /**
     * Creates an object with an cancel.gif image.
     */
    EditorModeSelectOrDragOrResizeOrFlipExpand() {
        super("cancel.gif");
    }

    /**
     * @see kiel.editor.controller.EditorMode
     * #mousePressed(java.awt.geom.Point2D)
     */
    public final void mousePressed(final Point2D p,
        final boolean isControlDown) {
        super.mousePressed(p, isControlDown);

        // For selection mode...
        automaticSelectionEnabled = !isControlDown;

        // This is done *here* because of performance issues
        if (isControlDown) {
            getEditor().updateMenuEdit();
        }

        // for dragging and resizing mode...
        getEditor().getGraph().getUndoClusterManager().setUndoClusterStart()
  ;

        selectedCells = getEditor().getGraph()
                .getSelectionModel().getSelectionCells();
        if (selectedCells != null) {
            selectedBounds = new Rectangle2D[selectedCells.length];
        } else {
            selectedBounds = null;
        }
        for (int i = 0; selectedCells != null
            && i < selectedCells.length; i++) {
```

```java
            selectedBounds[i] = (Rectangle2D) GraphConstants.getBounds(
                ((DefaultGraphCell) selectedCells[i]).getAttributes());
        }
    }

    /**
     *
     */
    private Object lastSelectedCell;

    /**
     * Handles automatic selection and updating menus on mouse moving events
     .
     * @see kiel.editor.controller.EditorMode
     * #mouseMoved(java.awt.geom.Point2D)
     */
    public final void mouseMoved(final Point2D p) {
        super.mouseMoved(p);

        Object selectedCell = getEditor().getGraph().getSelectionCell();
        if (lastSelectedCell != selectedCell) {
            lastSelectedCell = selectedCell;
            getEditor().updateMenuEdit();
        }

        // TODO Fix me! workaround: the cursor of EditorModeSelect...
        // will not be resetted after addDelimiter!
        if (getEditor() != null
            && (getEditor().getGraph().getCursor()
                == EditorModes.get(
                    EditorModeAddHorizontalDelimiter.class).getMyCursor())
               || getEditor().getGraph().getCursor()
               == EditorModes.get(
                   EditorModeAddVerticalDelimiter.class).getMyCursor()) {
            getEditor().getLogFile().log(2, "correct cursor");
            getEditor().getGraph().setCursor(getMyCursor());
        }

        // For selection mode
        if (!automaticSelectionEnabled
            || !(getEditor().getGraph().getModel() instanceof MyGraphModel)
            || getEditor().getGraph().isPopupVisible()) {

            return;
        }
        // select the node in which the mouse cursor is or, if exist, a
     child
        // of this node which intersects the mouse *area*.
        CellView toBeSelected = MyGraphModelUtilities.getSmallestViewOfType(
            CellView.class, new Rectangle((int) p.getX(),
                (int) p.getY(), 1, 1),
            getEditor().getGraph().getGraphLayoutCache(),
            false, null);
        if (toBeSelected == null) {
            toBeSelected = getEditor().getGraph().getGraphLayoutCache()
                .getMapping(getEditor().getGraph().getMyGraphModel()
                    .getRootCell(), false);
//          return;
        }
        Rectangle2D mouseRectangle = new AttributeMap.
    SerializableRectangle2D(
            p.getX() - Preferences.getSizeHandleSensitivity(),
            p.getY() - Preferences.getSizeHandleSensitivity(),
            Preferences.getSizeHandleSensitivity() * 2,
            Preferences.getSizeHandleSensitivity() * 2);

        for (int i = 0; i < toBeSelected.getChildViews().length; i++) {
            if (toBeSelected.getChildViews()[i] instanceof NodeView
                && toBeSelected.getChildViews()[i].getBounds().intersects(
                    mouseRectangle)) {
                toBeSelected = toBeSelected.getChildViews()[i];
                break;
            }
        }
        Object cell = toBeSelected.getCell();

        // Update Button States based on Current Selection
        EditorActions.get(ActionEditCopy.class).setEnabled(cell != null);
        EditorActions.get(ActionEditCut.class).setEnabled(cell != null);
        if (cell != null && cell != getEditor().getGraph().getSelectionCell
    ()) {
            getEditor().getGraph().getSelectionModel().setSelectionCell(cell
    );
            // update the collapsed checkbox items in the menubar and the
    popup
            // menu
            GraphicalObject go = MyGraphConstants.getGraphicalObject(
                ((DefaultGraphCell) cell).getAttributes());
            if (go instanceof ORState || go instanceof ANDState) {
                final boolean isCollapsed = MyGraphConstants.isCollapsed(
                    ((DefaultGraphCell) cell).getAttributes());

                getEditor().getMenuItemCollapsed().setSelected(isCollapsed);
                getEditor().getGraph().getMenuItemCollapsed().setSelected(
                    isCollapsed);
            }
        }
    }

    /**
     * @param cellClass .
     * @return .
     */
    private boolean isSelectedInstanceOf(final Class cellClass) {
        Object[] objects = getEditor().getGraph().getSelectionCells();
        for (int i = 0; objects != null && i < objects.length; i++) {
            if (objects[i] != null && cellClass.isInstance(objects[i])) {
                return true;
```

```
            }
        }
        return false;
    }

    /**
     *
     * @param cell .
     * @return .
     */
    private static Rectangle2D getPreviewBounds(final Object cell) {
        CellView view = getEditor().getGraph()
        .getGraphLayoutCache().getMapping(cell, false);
        if (view instanceof Previewable) {
            return ((Previewable) view).getPreviewBounds();
        } else if (cell != null) {
            getEditor().getLogFile().log(2, cell.getClass().getName());
        }
        return null;
    }

    /**
     *
     * @return .
     */
    private boolean isResizing() {

        CellView[] selectedViews = getEditor().getGraph()
            .getGraphLayoutCache().getMapping(selectedCells);

        for (int i = 0; selectedViews != null && i < selectedViews.length;
            i++) {

            if (selectedViews[i] instanceof Previewable
                && ((Previewable) selectedViews[i]).isResizing()) {

                return true;
            }
        }
        return false;
    }

    /**
     * @see kiel.editor.controller.EditorMode
     * #mouseDragged(java.awt.geom.Point2D)
     */
    public final void mouseDragged(final Point2D p) {
        super.mouseDragged(p);

        setModeEnabled(true);
        setIntersects(false);

        isDraggingOrResizing = true;

        if (isSelectedInstanceOf(RegionCell.class)) {
            setModeEnabled(false);
        } else if (isSelectedInstanceOf(DeepHistoryCell.class)) {
            setModeEnabled(false);
        } else if (isSelectedInstanceOf(DelimiterLineCell.class)
            && isResizing()) {
            setModeEnabled(false);
        } else if (isDraggingRootOrChangingUpperLeft()) {
            setModeEnabled(false);
        } else if (isSelectedInstanceOf(TransitionCell.class)) {
            setModeEnabled(true);
        } else if (isSelectedInstanceOf(PseudoStateCell.class)
            && isResizing()) {
            setModeEnabled(false);
        } else if (selectedCells != null) {
            boolean isResizing = isResizing();
            boolean intersects = false;

            if (isResizing) {

                Rectangle2D previewBounds = getPreviewBounds(selectedCells
[0])
                    .getBounds2D();

                if (selectedCells[0] instanceof ORStateCell
                    || selectedCells[0] instanceof ANDStateCell) {
                    // ensmall to the valid area
                    previewBounds.setRect(
                        previewBounds.getX(),
                        previewBounds.getY()
                            + Properties.getCompositeStateHeaderHeight(),
                        previewBounds.getWidth(),
                        previewBounds.getHeight()
                            - Properties.getCompositeStateHeaderHeight());
                }
                // and, ensmall to the intersections sensitivity...
                final int sens = Preferences.getIntersectionSensitivity();
                if (!(selectedCells[0] instanceof ANDStateCell)) { // make
                    previewBounds.setRect(// resizing ANDState possible!
                        previewBounds.getX() + sens,
                        previewBounds.getY() + sens,
                        previewBounds.getWidth() - 2 * sens,
                        previewBounds.getHeight() - 2 * sens);
                }
                intersects =
                    getEditor().getGraph().getMyGraphModel().intersect(
                        (DefaultGraphCell) selectedCells[0], previewBounds,
                        false, null, false)
                    || !getEditor().getGraph().getMyGraphModel()
                        .areAllChildrenInside((DefaultGraphCell)
                            selectedCells[0], previewBounds);

                // Check if all children are still in bounds
            } else { // IsMoving
```

```
            for (int cellCount = 0; cellCount < selectedCells.length;
                cellCount++) {

                boolean isRootCellDragged = false;
                for (int i = 0;
                    !isRootCellDragged && i < selectedCells.length; i++)
    {
                    if (getEditor().getGraph().getMyGraphModel()
                        .getRootCell() == selectedCells[i]) {
                        isRootCellDragged = true;
                    }
                }
                Rectangle2D previewBounds = getPreviewBounds(
                    selectedCells[cellCount]);

                if (isRootCellDragged) {
                    intersects = true;
                } else if (previewBounds != null) {
                    // Is valid for move only!!
                    intersects = intersects
                    || getEditor().getGraph().getMyGraphModel()
                        .intersect((DefaultGraphCell)
                        selectedCells[cellCount], previewBounds, true,
                        Arrays.asList(selectedCells), false);
                } else {
                    getEditor().getLogFile().log(2, getClass()
                        + ".mouseDragged(): previewBounds == null");
                }
            }
        }
        setIntersects(intersects);
    }
}

/**
 * .
 * @return .
 */
private boolean isDraggingRootOrChangingUpperLeft() {

    DefaultGraphCell root =
        getEditor().getGraph().getMyGraphModel().getRootCell();
    return
        getEditor().getGraph().getSelectionCells() != null
     && getEditor().getGraph().getSelectionCells().length > 0
     && root == getEditor().getGraph().getSelectionCells()[0]
     && getPreviewBounds(root) != null
     && (getPreviewBounds(root).getX()
        != Preferences.getInitialGraphOffset()
        || getPreviewBounds(root).getY()
        != Preferences.getInitialGraphOffset());
}

// @TODO
/**
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
 a>
 */
private static final class EditorModeDrag
    extends EditorModeSelectOrDragOrResizeOrFlipExpand { }

// @TODO
/**
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
 a>
 */
private static final class EditorModeResize
    extends EditorModeSelectOrDragOrResizeOrFlipExpand { }

/**
 * @see kiel.editor.controller.EditorMode
 * #mouseReleased(java.awt.geom.Point2D)
 */
public final void mouseReleased(final Point2D p) {
    super.mouseReleased(p);

    for (int i = 0; selectedCells != null
        && i < selectedCells.length; i++) {
        // has to be done here, since this is independent from dragging
 or
        // resizing in case of adding/deleting transition points
        CellView view = getEditor().getGraph().getGraphLayoutCache()
            .getMapping(selectedCells[i], false);
        Point2D labelPosition = null;
        if (view != null) {
            labelPosition = GraphConstants.getLabelPosition(
                view.getAttributes());
        }
        if (selectedCells[i] instanceof TransitionCell) {
            // or source or target have been moved!!!!!
            getEditor().getGraph().getMyGraphModel()
                .updateKielTransitionLabelsPosition(
                    (TransitionCell) selectedCells[i],
                    labelPosition);
        } else if (selectedCells[i] instanceof NodeCell) {
            Iterator edges = DefaultGraphModel.getEdges(
                getEditor().getGraph().getModel(),
                new Object[]{selectedCells[i]}).iterator();
            while (edges.hasNext()) {
                getEditor().getGraph().getMyGraphModel()
                    .updateKielTransitionLabelsPosition(
                        (TransitionCell) edges.next(),
                        labelPosition);
            }
        }
    }

    if (isFlipExpandCompositeStateTrigger(p)) {
```

```
                EditorActions.get(ActionEditCollapsed.class).actionPerformed(
null);
   } else if (isDraggingOrResizing) {

       if (isModeEnabled() && !isIntersects()) {
           for (int i = 0; selectedCells != null
               && i < selectedCells.length; i++) {

               if (selectedCells[i] instanceof DelimiterLineCell) {
                   getEditor().getGraph().getMyGraphModel()
                       .delimiterLineCellMoved(
                           (DelimiterLineCell) selectedCells[i],
                           selectedBounds[i],
                           getEditor().getGraph().getGraphLayoutCache()
);
                   continue;
               }
               if (selectedCells[i] instanceof ANDStateCell) {
                   getEditor().getGraph().getMyGraphModel()
                       .andStateCellEnlarged(
                           (ANDStateCell) selectedCells[i],
                           selectedBounds[i]);
               }
               // set the parent of the dragged/resized cell
               getEditor().getGraph().getMyGraphModel().changeParent(
                   (DefaultGraphCell) selectedCells[i],
                   getEditor().getGraph().getGraphLayoutCache());

               // set the parent of all cells and transitions
               // that are now inside the dragged/resized cell
               CellView view = getEditor().getGraph()
                   .getGraphLayoutCache().getMapping(
                       selectedCells[i], false);
               if (view == null) {
                   System.out.println("EditorModeSelectOrDragOrResizeOr
"
                           + "FlipExpand: view is null");
                   continue;
               }
               Object[] newChildren = getEditor().getGraph()
                   .getMyGraphModel().getAllCells(view.getBounds());
               for (int newCount = 0; newCount < newChildren.length;
                   newCount++) {
                   if (((DefaultGraphCell) newChildren[newCount])
                           .getParent()
                       == ((DefaultGraphCell) selectedCells[i])
                           .getParent()) {
                       getEditor().getGraph().getMyGraphModel()
                       .changeParent(
                           (DefaultGraphCell) newChildren[newCount],
                           getEditor().getGraph().getGraphLayoutCache()
);
                       Object[] edges = MyGraphModel.getEdges(
                           getEditor().getGraph().getModel(),
                           newChildren[newCount],
                           false);
                       for (int e = 0; e < edges.length; e++) {
                           getEditor().getGraph().getMyGraphModel()
                               .changeParent(
                                   (DefaultGraphCell) edges[e],
                                   getEditor().getGraph()
                                   .getGraphLayoutCache());
                       }
                   }
               }
           }
       }
       if (isDraggingOrResizing) {
           getEditor().getGraph().getUndoClusterManager()
               .setUndoClusterEnd(new EditorModeDrag());
       } else {
           getEditor().getGraph().getUndoClusterManager()
               .setUndoClusterEnd(new EditorModeResize());
       }
       if (!(isModeEnabled() && !isIntersects())) {
           getEditor().getGraph().getUndoClusterManager()
               .nonRedoableUndo();
       }
       selectedCells = null;
   }
   isDraggingOrResizing = false;
   EditorModes.get(EditorModeSelectOrDragOrResizeOrFlipExpand.class)
       .start();
}

/**
 *
 * @param p .
 * @return .
 */
private boolean isFlipExpandCompositeStateTrigger(final Point2D p) {

    if (getEditor().getGraph().getMyGraphModel() == null) {
        return false;
    }

    CompositeStateCell cell = (CompositeStateCell) getEditor()
        .getGraph().getMyGraphModel()
        .getSmallestCellOfType(CompositeStateCell.class,
            new AttributeMap.SerializableRectangle2D(
                p.getX(), p.getY(), 1, 1), false);

    if (cell != null && (!(cell instanceof RegionCell))) {
        Rectangle2D bounds = GraphConstants.getBounds(
            cell.getAttributes()).getBounds2D();

        if (bounds.getX() + Preferences.getFlipArea().x < p.getX()
            &&  p.getX() < bounds.getX() + Preferences.getFlipArea().x
```

```java
                + Preferences.getFlipArea().width
                &&  bounds.getY() + Preferences.getFlipArea().y < p.getY()
                &&  p.getY() < bounds.getY() + Preferences.getFlipArea().y
                + Preferences.getFlipArea().height) {

                    return true;
            }
        }
        return false;
    }

    /**
     *
     * @see kiel.editor.MorphingLayouter.MyListener#morphingFinished()
     */
    public final void morphingFinished() {
        getEditor().getGraph().getUndoClusterManager().setUndoClusterEnd(
      this);
    }

    /**
     * @see kiel.editor.controller.EditorMode#start()
     */
    public final void start() {
        super.start();
        getEditor().getGraph().setCursor(getMyCursor());
    }

    /**
     * @see kiel.editor.controller.EditorMode#done()
     */
    public void done() {
    }
}
```

kiel.editor.controller.MyMarqueeHandler

```java
package kiel.editor.controller;

import java.awt.Color;
import java.awt.Cursor;
import java.awt.Graphics;
import java.awt.event.MouseEvent;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;

import kiel.editor.MyGraphModel;

import org.jgraph.JGraph;
import org.jgraph.graph.BasicMarqueeHandler;
import org.jgraph.graph.GraphConstants;
import org.jgraph.graph.Port;
import org.jgraph.graph.PortView;

/**
 * A simple implementation of a marquee handler for JGraph.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.15 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class MyMarqueeHandler extends BasicMarqueeHandler {

    /**
     * Holds the Start and the Current Point.
     */
    private Point2D start, current;

    /**
     * Holds the First and the Current Port.
     */
    private PortView port, firstPort;

    /**
     *
     */
    private JGraph graph;

    /**
     * @param aGraph .
     *
     */
    public MyMarqueeHandler(final JGraph aGraph) {
        graph = aGraph;
    }

    /**
     * Override to Gain Control (for PopupMenu and ConnectMode).
     * @see org.jgraph.graph.BasicMarqueeHandler#isForceMarqueeEvent(
     * java.awt.event.MouseEvent)
     */
    public boolean isForceMarqueeEvent(final MouseEvent e) {
        if (e.isShiftDown()) {
            return false;
        }
        // If Right Mouse Button we want to Display the PopupMenu
        if (e.isPopupTrigger()) {
            // Return Immediately
            return true;
        }
        // Find and Remember Port
        port = getSourcePortAt(e.getPoint());
        // If Port Found and in ConnectMode (=Ports Visible)
        if (port != null && graph.isPortsVisible()) {
            return true;
        }
        // Else Call Superclass
        return super.isForceMarqueeEvent(e);
    }

    /**
     * Display PopupMenu or Remember Start Location and First Port.
     * @see org.jgraph.graph.BasicMarqueeHandler#mousePressed(
     * java.awt.event.MouseEvent)
     */
    public void mousePressed(final MouseEvent e) {
        if (e.isPopupTrigger()) {
            return;
        // Else if in ConnectMode and Remembered Port is Valid
        } else if (port != null && graph.isPortsVisible()) {
            // Remember Start Location
            start = graph.toScreen(port.getLocation(null));
            // Remember First Port
            firstPort = port;
        } else {
            // Call Superclass
            super.mousePressed(e);
        }
    }

    /**
     * Find Port under Mouse and Repaint Connector.
     * @see org.jgraph.graph.BasicMarqueeHandler#mouseDragged(
     * java.awt.event.MouseEvent)
     */
    public void mouseDragged(final MouseEvent e) {
        // If remembered Start Point is Valid
        if (start != null) {
            // Fetch Graphics from Graph
```

```java
            Graphics g = graph.getGraphics();
            // Reset Remembered Port
            PortView newPort = getTargetPortAt(e.getPoint());
            // Do not flicker (repaint only on real changes)
            if (newPort == null || newPort != port) {
                // Xor-Paint the old Connector (Hide old Connector)
                paintConnector(Color.black, graph.getBackground(), g);
                // If Port was found then Point to Port Location
                port = newPort;
                if (port != null) {
                    current = graph.toScreen(port.getLocation(null));
                // Else If no Port was found then Point to Mouse Location
                } else {
                    current = graph.snap(e.getPoint());
                }
                // Xor-Paint the new Connector
                paintConnector(graph.getBackground(), Color.black, g);
            }
        }
        // Call Superclass
        super.mouseDragged(e);
    }

    /**
     *
     * @param point .
     * @return .
     */
    protected PortView getSourcePortAt(final Point2D point) {
        // Disable jumping
        graph.setJumpToDefaultPort(false);
        PortView result;
        try {
            // Find a Port View in Model Coordinates and Remember
            result = graph.getPortViewAt(point.getX(), point.getY());
        } finally {
            graph.setJumpToDefaultPort(true);
        }
        return result;
    }

    /**
     * Find a Cell at point and Return its first Port as a PortView.
     * @param point .
     * @return .
     */
    protected PortView getTargetPortAt(final Point2D point) {
        // Find a Port View in Model Coordinates and Remember
        return graph.getPortViewAt(point.getX(), point.getY());
    }

    /**
     * Connect the First Port and the Current Port in the Graph or Repaint.
     * @see org.jgraph.graph.BasicMarqueeHandler#mouseReleased(
     * java.awt.event.MouseEvent)
     */
    public void mouseReleased(final MouseEvent e) {
        if (e.isPopupTrigger() && !e.isShiftDown()) {
            return;
        }
        // If Valid Event, Current and First Port
        if (e != null
            && port != null
            && firstPort != null
            && firstPort != port) {
            // Then Establish Connection
            try {
                ((MyGraphModel) graph.getModel()).addTransition(
                    (Port) firstPort.getCell(), (Port) port.getCell(),
                    EditorAction.getEditor().getCurrentTransitionType(),
                    EditorAction.getEditor().getCurrentTransitionLine(),
                    null);
            } catch (Exception ex) {
                EditorAction.getEditor().getKielFrame().handleException(
                    ex.getMessage(), false, ex);
            }

        // Else Repaint the Graph
        } else {
            graph.repaint();
        }
        // Reset Global Vars
        firstPort = null;
        port = null;
        start = null;
        current = null;
        // Call Superclass
        super.mouseReleased(e);
    }

    /**
     * Show Special Cursor if Over Port.
     * @see org.jgraph.graph.BasicMarqueeHandler#mouseMoved(
     * java.awt.event.MouseEvent)
     */
    public void mouseMoved(final MouseEvent e) {
        // Check Mode and Find Port
        if (e != null
            && getSourcePortAt(e.getPoint()) != null
            && graph.isPortsVisible()) {
            // Set Cusor on Graph (Automatically Reset)
            graph.setCursor(new Cursor(Cursor.HAND_CURSOR));
            // Consume Event
            // Note: This is to signal the BasicGraphUI's
            // MouseHandle to stop further event processing.
            e.consume();
        } else {
            // Call Superclass
```

```java
            super.mouseMoved(e);
        }
    }

    /**
     * Use Xor-Mode on Graphics to Paint Connector.
     * @param fg .
     * @param bg .
     * @param g .
     */
    protected void paintConnector(final Color fg, final Color bg,
        final Graphics g) {
        // Set Foreground
        g.setColor(fg);
        // Set Xor-Mode Color
        g.setXORMode(bg);
        // Highlight the Current Port
        paintPort(graph.getGraphics());
        // If Valid First Port, Start and Current Point
        if (firstPort != null && start != null && current != null) {
            // Then Draw A Line From Start to Current Point
            g.drawLine((int) start.getX(),
                       (int) start.getY(),
                       (int) current.getX(),
                       (int) current.getY());
        }
    }

    /**
     * Use the Preview Flag to Draw a Highlighted Port.
     * @param g .
     */
    protected void paintPort(final Graphics g) {
        // If Current Port is Valid
        if (port != null) {
            // If Not Floating Port...
            // ...Then use Parent's Bounds
            Rectangle2D r;
            if (GraphConstants.getOffset(port.getAttributes()) != null) {
                r = port.getBounds();
            } else {
                r = port.getParentView().getBounds();
            }
            // Scale from Model to Screen
            r = graph.toScreen((Rectangle2D) r.clone());
            // Add Space For the Highlight Border
            final int border = 3;
            r.setFrame(
                r.getX() - border,
                r.getY() - border,
                r.getWidth() + border * 2,
                r.getHeight() + border * 2);
            // Paint Port in Preview (=Highlight) Mode
            graph.getUI().paintCell(g, port, r, true);
        }
    }
}
```

kiel.editor.graph.ANDStateCell

```
package kiel.editor.graph;

/**
 * Describes an ANDState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class ANDStateCell extends CompositeStateCell {

    /**
     * @param userObject .
     */
    public ANDStateCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.AndStateRenderer

```java
package kiel.editor.graph;

import java.awt.Graphics;
import java.awt.Graphics2D;

import kiel.graphicalInformations.Properties;

import org.jgraph.graph.GraphConstants;

/**
 * Describes an ANDState in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/04/01 12:26:33 $
 */
class AndStateRenderer extends CompositeStateRenderer {

    /**
     * @see org.jgraph.graph.VertexRenderer#paintSelectionBorder(
     * java.awt.Graphics)
     */
    protected void paintSelectionBorder(final Graphics g) {
        super.paintSelectionBorder(g);
        ((Graphics2D) g).setStroke(getGraph().getStroke(getView()));
        g.drawLine(0, Properties.getCompositeStateHeaderHeight(),
            getSize().width - 2,
            Properties.getCompositeStateHeaderHeight());
        ((Graphics2D) g).setStroke(GraphConstants.SELECTION_STROKE);
    }
}
```

kiel.editor.graph.ANDStateView

```java
package kiel.editor.graph;

/**
 * Describes an ANDState in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.21 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class ANDStateView extends CompositeStateView {

    /**
     * @param cell .
     */
    public ANDStateView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.CellViewRendererFactory

```java
package kiel.editor.graph;

import java.util.HashMap;
import java.util.Hashtable;

import org.jgraph.graph.CellViewRenderer;

/**
 * This class defines all the drawing information of the CellView
 * classes representing the diagram elements on the view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.21 $ last modified $Date: 2005/07/25 11:44:37 $
 */
public final class CellViewRendererFactory {

    /**
     *
     */
    private static final HashMap RENDERER_MAPPING = new HashMap();

    static {
        RENDERER_MAPPING.put(
            ANDStateView.class, AndStateRenderer.class);
        RENDERER_MAPPING.put(
            ChoiceView.class, ChoiceRenderer.class);
        RENDERER_MAPPING.put(
            CompositeStateView.class, CompositeStateRenderer.class);
        RENDERER_MAPPING.put(
            DeepHistoryView.class, DeepHistoryRenderer.class);
        RENDERER_MAPPING.put(
            DynamicChoiceView.class, DynamicChoiceRenderer.class);
        RENDERER_MAPPING.put(
            FinalANDStateView.class, FinalAndStateRenderer.class);
        RENDERER_MAPPING.put(
            FinalORStateView.class, FinalOrStateRenderer.class);
        RENDERER_MAPPING.put(
            FinalSimpleStateView.class, FinalSimpleStateRenderer.class);
        RENDERER_MAPPING.put(
            FinalStateView.class, FinalStateRenderer.class);
        RENDERER_MAPPING.put(
            HistoryView.class, HistoryRenderer.class);
        RENDERER_MAPPING.put(
            InitialStateView.class, InitialStateRenderer.class);
        RENDERER_MAPPING.put(
            NodeView.class, NodeRenderer.class);
        RENDERER_MAPPING.put(
            NormalTerminationView.class, NormalTerminationRenderer.class);
        RENDERER_MAPPING.put(
            ORStateView.class, OrStateRenderer.class);
        RENDERER_MAPPING.put(
            PseudoStateView.class, PseudoStateRenderer.class);
        RENDERER_MAPPING.put(
            RegionView.class, RegionRenderer.class);
        RENDERER_MAPPING.put(
            StateView.class, StateRenderer.class);
        RENDERER_MAPPING.put(
            StrongAbortionView.class, StrongAbortionRenderer.class);
        RENDERER_MAPPING.put(
            SuspendView.class, SuspendRenderer.class);
        RENDERER_MAPPING.put(
            TransitionView.class, TransitionRenderer.class);
        RENDERER_MAPPING.put(
            DelimiterLineView.class, DelimiterLineRenderer.class);
    }

    /**
     *
     *
     */
    private CellViewRendererFactory() { }

    /**
     * Comment for <code>cache</code>.
     */
    private static Hashtable cache = new Hashtable();

    /**
     * @param aViewClass The CellView class
     * @return The renderer class
     */
    static CellViewRenderer get(final Class aViewClass) {

        Class viewClass = aViewClass;
        Class rendererClass = null;
        while (rendererClass == null) {
            rendererClass = (Class) RENDERER_MAPPING.get(viewClass);
            viewClass = viewClass.getSuperclass();
        }

        CellViewRenderer instance = (CellViewRenderer) cache.get(
    rendererClass);
        if (instance == null) {
            try {
                instance = (CellViewRenderer) rendererClass.newInstance();
                cache.put(rendererClass, instance);
            } catch (InstantiationException e) {
                e.printStackTrace();
            } catch (IllegalAccessException e) {
                e.printStackTrace();
            }
        }
        return instance;
    }
}
```

kiel.editor.graph.ChoiceCell

```java
package kiel.editor.graph;

/**
 * Describes a Choice in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class ChoiceCell extends PseudoStateCell {

    /**
     * @param userObject .
     */
    public ChoiceCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.ChoiceRenderer

```java
package kiel.editor.graph;

/**
 * Describes a Choice in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.7 $ last modified $Date: 2005/03/15 11:52:53 $
 */
class ChoiceRenderer extends PseudoStateRenderer {

    /**
     * @see kiel.editor.graph.PseudoStateRenderer#getMyName()
     */
    protected String getMyName() {
        return "C";
//          image = ResourceLoader.get("DynamicChoiceGrey.gif").getImage();
    }
}
```

kiel.editor.graph.ChoiceView

```
package kiel.editor.graph;

/**
 * Describes a Choice in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.17 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class ChoiceView extends PseudoStateView {

    /**
     * @param cell .
     */
    public ChoiceView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.CompositeStateCell

```java
package kiel.editor.graph;

/**
 * Describes a CompositeState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class CompositeStateCell extends StateCell {

    /**
     * @param userObject .
     */
    public CompositeStateCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.CompositeStateRenderer

```java
package kiel.editor.graph;

import java.awt.Graphics;

import kiel.dataStructure.CompositeState;
import kiel.editor.resources.Preferences;
import kiel.editor.resources.ResourceBundle;
import kiel.graphicalInformations.Properties;

import org.jgraph.graph.DefaultGraphCell;


/**
 * Describes a CompositeState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/07/25 11:44:37 $
 */
class CompositeStateRenderer extends StateRenderer {

    /**
     *
     * @see org.jgraph.graph.VertexRenderer#paintSelectionBorder(
     * java.awt.Graphics)
     */
    protected void paintSelectionBorder(final Graphics g) {
        super.paintSelectionBorder(g);
        boolean isCollapsed = MyGraphConstants.isCollapsed(
            ((DefaultGraphCell) getView().getCell()).getAttributes());
        if (Preferences.getShowCollapseButtons()) {
            if (isCollapsed) {
                g.drawImage(
                    getGraph().getPlusImage(),
                    Preferences.getCompositeStateImageOffset(),
                    Preferences.getCompositeStateImageOffset(),
                    null);
            } else {
                g.drawImage(
                    getGraph().getMinusImage(),
                    Preferences.getCompositeStateImageOffset(),
                    Preferences.getCompositeStateImageOffset(),
                    null);
            }
        }
    }

    /**
     * .
     * @see kiel.editor.graph.StateRenderer#drawVariablesAndEvents(
     * java.awt.Graphics, int)
     */
    protected int drawVariablesAndEvents(final Graphics g,
        final int initialYOffset) {

        int yOffset = initialYOffset
        + Properties.getCompositeStateHeaderHeight();

        if (Preferences.getShowStateActions()) {
            CompositeState state = (CompositeState) MyGraphConstants
                .getGraphicalObject(((DefaultGraphCell) getView().getCell())
                .getAttributes());
            if (Preferences.getShowCompositeStateVariables()) {
                yOffset = draw(ResourceBundle.getString("Variables")
                    + ": ", state.getVariables(), yOffset, g);
            }
            if (Preferences.getShowCompositeStateEvents()) {
                yOffset = draw(ResourceBundle.getString("Events")
                    + ": ", state.getLocalEvents(), yOffset, g);
            }
        }
        return yOffset;
    }
}
```

kiel.editor.graph.CompositeStateView

```
package kiel.editor.graph;

/**
 * Describes a CompositeState in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.19 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class CompositeStateView extends StateView {

    /**
     * @param cell .
     */
    public CompositeStateView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.ConditionalTransitionCell

```java
package kiel.editor.graph;

/**
 * Describes a ConditionalTransition in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class ConditionalTransitionCell extends TransitionCell {

    /**
     *
     */
    public ConditionalTransitionCell() {
        super();
    }

    /**
     * @param userObject .
     */
    public ConditionalTransitionCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.ConditionalTransitionView

```java
package kiel.editor.graph;

/**
 * Describes a ConditionalTransition in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.14 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class ConditionalTransitionView extends TransitionView {

    /**
     * @param cell .
     */
    public ConditionalTransitionView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.DeepHistoryCell

```
package kiel.editor.graph;

/**
 * Describes a DeepHistory in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class DeepHistoryCell extends PseudoStateCell {

    /**
     * @param userObject .
     */
    public DeepHistoryCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.DeepHistoryRenderer

```
package kiel.editor.graph;

/**
 * Describes a DeepHistory in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.7 $ last modified $Date: 2005/03/15 11:52:54 $
 */
class DeepHistoryRenderer extends PseudoStateRenderer {

    /**
     * @see kiel.editor.graph.PseudoStateRenderer#getMyName()
     */
    protected String getMyName() {
        return "H*";
//          image = ResourceLoader.get("DeepHistoryGrey.gif").getImage();
    }
}
```

kiel.editor.graph.DeepHistoryView

```java
package kiel.editor.graph;

/**
 * Describes a DeepHistory in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.17 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class DeepHistoryView extends PseudoStateView {

    /**
     * @param cell .
     */
    public DeepHistoryView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.DelimiterLineCell

```
package kiel.editor.graph;

/**
 * Describes an ANDState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.5 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class DelimiterLineCell extends GraphicalObjectCell {

    /**
     * @param userObject .
     */
    public DelimiterLineCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.DelimiterLineRenderer

```java
package kiel.editor.graph;

import java.awt.Graphics;
import java.awt.Graphics2D;

import kiel.editor.MyJGraph;

import org.jgraph.graph.GraphConstants;
import org.jgraph.graph.VertexRenderer;

/**
 * Describes an ANDState in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.7 $ last modified $Date: 2005/07/24 20:00:30 $
 */
class DelimiterLineRenderer extends VertexRenderer {

    /**
     * @see org.jgraph.graph.VertexRenderer#paintSelectionBorder(
     * java.awt.Graphics)
     */
    protected void paintSelectionBorder(final Graphics g) {
        super.paintSelectionBorder(g);
        g.setColor(((MyJGraph) graph).getViewsBlack(view));
        ((Graphics2D) g).setStroke(GraphConstants.SELECTION_STROKE);
//        System.out.println(getBounds());
        if (getBounds().width < getBounds().height) {
            g.drawLine(0, 0, 0, getBounds().height);
        } else {
            g.drawLine(0, 0, getBounds().width, 0);
        }
    }
}
```

kiel.editor.graph.DelimiterLineView

```java
package kiel.editor.graph;

import java.awt.geom.Rectangle2D;

import org.jgraph.graph.CellHandle;
import org.jgraph.graph.CellViewRenderer;
import org.jgraph.graph.GraphConstants;
import org.jgraph.graph.GraphContext;
import org.jgraph.graph.VertexView;

/**
 * Describes an ANDState in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/07/25 11:44:37 $
 */
public class DelimiterLineView extends VertexView implements Previewable {

    /**
     *
     * @see org.jgraph.graph.AbstractCellView#getRenderer()
     */
   public final CellViewRenderer getRenderer() {
       return CellViewRendererFactory.get(getClass());
   }

    /**
     * @param cell .
     */
    public DelimiterLineView(final Object cell) {
        super(cell);
    }

    /**
     * Comment for <code>isResizing</code>.
     */
    private boolean isResizing;

    /**
     * .
     * @param aBounds .
     */
    public final void setPreviewBounds(final Rectangle2D aBounds) {
        previewBounds = aBounds;
    }

    /**
     * .
     * @param b .
     */
    public final void setResizing(final boolean b) {
        isResizing = b;
    }

    /**
     * @return .
     */
    public final boolean isResizing() {
        return isResizing;
    }

    /**
     * @see org.jgraph.graph.CellView#getHandle(org.jgraph.graph.
     GraphContext)
     */
    public final CellHandle getHandle(final GraphContext context) {
        if (GraphConstants.isSizeable(getAllAttributes())
                && context.getGraph().isSizeable()) {

            return new MySizeHandle(this, context);
        }
        return null;
    }

    /**
     *
     */
    private Rectangle2D previewBounds;

    /**
     * .
     * @return .
     */
    public final Rectangle2D getPreviewBounds() {
        return previewBounds;
    }
}
```

kiel.editor.graph.DynamicChoiceCell

```java
package kiel.editor.graph;

/**
 * Describes a DynamicChoice in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class DynamicChoiceCell extends PseudoStateCell {

    /**
     * @param userObject .
     */
    public DynamicChoiceCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.DynamicChoiceRenderer

```
package kiel.editor.graph;

/**
 * Describes a DynamicChoice in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.7 $ last modified $Date: 2005/03/15 11:52:53 $
 */
class DynamicChoiceRenderer extends PseudoStateRenderer {

    /**
     * @see kiel.editor.graph.PseudoStateRenderer#getMyName()
     */
    protected String getMyName() {
        return "C";
//          image = ResourceLoader.get("DynamicChoiceGrey.gif").getImage();
    }
}
```

kiel.editor.graph.DynamicChoiceView

```
package kiel.editor.graph;

/**
 * Describes a DynamicChoice in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.17 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class DynamicChoiceView extends PseudoStateView {

    /**
     * @param cell .
     */
    public DynamicChoiceView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.EdgeCell

```
package kiel.editor.graph;

import org.jgraph.graph.DefaultEdge;

/**
 * Describes an Edge in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public abstract class EdgeCell extends DefaultEdge {

    /**
     *
     */
    public EdgeCell() {
        super();
    }

    /**
     * @param userObject .
     */
    public EdgeCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.EdgeView

```java
package kiel.editor.graph;

import java.awt.Graphics;
import java.awt.event.MouseEvent;

import javax.swing.SwingUtilities;

import kiel.editor.resources.Preferences;

import org.jgraph.graph.CellHandle;
import org.jgraph.graph.CellViewRenderer;
import org.jgraph.graph.GraphContext;

/**
 * Describes an Edge in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.18 $ last modified $Date: 2005/07/15 20:25:02 $
 */
public abstract class EdgeView extends org.jgraph.graph.EdgeView {

    /**
     *
     * @see org.jgraph.graph.AbstractCellView#getRenderer()
     */
    public CellViewRenderer getRenderer() {
        return CellViewRendererFactory.get(getClass());
    }

    /**
     * @see org.jgraph.graph.CellView#getHandle(org.jgraph.graph.
     GraphContext)
     */
    public final CellHandle getHandle(final GraphContext context) {
        return new MyEdgeHandle(this, context);
    }

    /**
     * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
     a>
     */
    private static class MyEdgeHandle extends EdgeHandle {

        /**
         * @param edge .
         * @param ctx .
         */
        public MyEdgeHandle(final org.jgraph.graph.EdgeView edge,
            final GraphContext ctx) {
            super(edge, ctx);
        }

        /**
         * @see org.jgraph.graph.CellHandle#paint(java.awt.Graphics)
         */
        public void paint(final Graphics g) {
            if (Preferences.getSizeHandleVisible()) {
                super.paint(g);
            }
        }

        /**
         * @see org.jgraph.graph.EdgeView.EdgeHandle#isAddPointEvent(
         * java.awt.event.MouseEvent)
         */
        public boolean isAddPointEvent(final MouseEvent event) {
            return SwingUtilities.isRightMouseButton(event)
                && event.isShiftDown();
        }

        /**
         * @see org.jgraph.graph.EdgeView.EdgeHandle#isRemovePointEvent(
         * java.awt.event.MouseEvent)
         */
        public boolean isRemovePointEvent(final MouseEvent event) {
            return SwingUtilities.isRightMouseButton(event)
                && event.isShiftDown();
        }
    }

    /**
     * @param cell .
     */
    public EdgeView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.FinalANDStateCell

```
package kiel.editor.graph;

/**
 * Describes a FinalANDState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class FinalANDStateCell extends ANDStateCell {

    /**
     * @param userObject .
     */
    public FinalANDStateCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.FinalANDStateRenderer

```java
package kiel.editor.graph;

import java.awt.Graphics;
import java.awt.Graphics2D;

import kiel.editor.resources.Preferences;

/**
 * Describes a FinalANDState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.10 $ last modified $Date: 2005/04/01 12:26:33 $
 */
class FinalAndStateRenderer extends AndStateRenderer {

    /**
     * @see org.jgraph.graph.VertexRenderer#paintSelectionBorder(
     * java.awt.Graphics)
     */
    protected void paintSelectionBorder(final Graphics g) {
        super.paintSelectionBorder(g);
        ((Graphics2D) g).setStroke(getGraph().getStroke(getView()));
        g.drawRoundRect(
            Preferences.getFinalStateBorder(),
            Preferences.getFinalStateBorder(),
            getSize().width - (Preferences.getFinalStateBorder() * 2 + 1),
            getSize().height - (Preferences.getFinalStateBorder() * 2 + 1),
            Preferences.getArcWidthHeight(),
            Preferences.getArcWidthHeight());
    }
}
```

kiel.editor.graph.FinalANDStateView

```
package kiel.editor.graph;

/**
 * Describes a FinalANDState in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.15 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class FinalANDStateView extends ANDStateView {

    /**
     * @param cell .
     */
    public FinalANDStateView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.FinalORStateCell

```java
package kiel.editor.graph;

/**
 * Describes a FinalORState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class FinalORStateCell extends ORStateCell {

    /**
     * @param userObject .
     */
    public FinalORStateCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.FinalORStateRenderer

```
package kiel.editor.graph;

import java.awt.Graphics;

import kiel.editor.resources.Preferences;

/**
 * Describes a FinalORState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.7 $ last modified $Date: 2005/03/15 11:52:54 $
 */
class FinalOrStateRenderer extends OrStateRenderer {

    /**
     * @see org.jgraph.graph.VertexRenderer#paintSelectionBorder(
     * java.awt.Graphics)
     */
    protected void paintSelectionBorder(final Graphics g) {
        super.paintSelectionBorder(g);
        g.drawRoundRect(
            Preferences.getFinalStateBorder(),
            Preferences.getFinalStateBorder(),
            getSize().width - (Preferences.getFinalStateBorder() * 2 + 1),
            getSize().height - (Preferences.getFinalStateBorder() * 2 + 1),
            Preferences.getArcWidthHeight(),
            Preferences.getArcWidthHeight());
    }
}
```

kiel.editor.graph.FinalORStateView

```
package kiel.editor.graph;

/**
 * Describes a FinalORState in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.15 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class FinalORStateView extends ORStateView {

    /**
     * @param cell .
     */
    public FinalORStateView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.FinalSimpleStateCell

```
package kiel.editor.graph;

/**
 * Describes a FinalSimpleState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class FinalSimpleStateCell extends SimpleStateCell {

    /**
     * @param userObject .
     */
    public FinalSimpleStateCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.FinalSimpleStateRenderer

```
package kiel.editor.graph;

import java.awt.Graphics;
import java.awt.Graphics2D;

import kiel.editor.resources.Preferences;

/**
 * Describes a FinalSimpleState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.10 $ last modified $Date: 2005/04/01 12:26:33 $
 */
class FinalSimpleStateRenderer extends StateRenderer {

    /**
     * @see org.jgraph.graph.VertexRenderer#paintSelectionBorder(
     * java.awt.Graphics)
     */
    protected void paintSelectionBorder(final Graphics g) {
        ((Graphics2D) g).setStroke(getGraph().getStroke(getView()));
        g.setColor(getGraph().getViewsBlack(getView()));
        g.drawOval(0, 0, getSize().width - 1, getSize().height - 1);
        g.drawOval(
            Preferences.getFinalStateBorder(),
            Preferences.getFinalStateBorder(),
            getSize().width - (Preferences.getFinalStateBorder() * 2 + 1),
            getSize().height - (Preferences.getFinalStateBorder() * 2 + 1));
        final int initialYOffset = 30;
        drawActions(g, initialYOffset);
    }
}
```

kiel.editor.graph.FinalSimpleStateView

```
package kiel.editor.graph;

/**
 * Describes a FinalSimpleState in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.17 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class FinalSimpleStateView extends SimpleStateView {

    /**
     * @param cell .
     */
    public FinalSimpleStateView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.FinalStateCell

```
package kiel.editor.graph;

/**
 * Describes a FinalState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class FinalStateCell extends StateCell {

    /**
     * @param userObject .
     */
    public FinalStateCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.FinalStateRenderer

```java
package kiel.editor.graph;

import java.awt.Graphics;
import java.awt.Graphics2D;

import kiel.editor.resources.Preferences;

/**
 * Describes a FinalState in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.10 $ last modified $Date: 2005/04/01 12:26:33 $
 */
class FinalStateRenderer extends StateRenderer {

    /**
     * @see org.jgraph.graph.VertexRenderer#paintSelectionBorder(
     * java.awt.Graphics)
     */
    protected void paintSelectionBorder(final Graphics g) {
        ((Graphics2D) g).setStroke(getGraph().getStroke(getView()));
        g.setColor(getGraph().getViewsBlack(getView()));
        g.drawOval(0, 0, getSize().width - 1, getSize().height - 1);
        g.drawOval(
            Preferences.getFinalStateBorder(),
            Preferences.getFinalStateBorder(),
            getSize().width - (Preferences.getFinalStateBorder() * 2 + 1),
            getSize().height - (Preferences.getFinalStateBorder() * 2 + 1));
        final int initialYOffset = 30;
        drawActions(g, initialYOffset);
    }
}
```

kiel.editor.graph.FinalStateView

```
package kiel.editor.graph;

/**
 * Describes a FinalState in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.18 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class FinalStateView extends StateView {

    /**
     * @param cell .
     */
    public FinalStateView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.ForkConnectorCell

```
package kiel.editor.graph;

/**
 * Describes a ForkConnector in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class ForkConnectorCell extends PseudoStateCell {

    /**
     * @param userObject .
     */
    public ForkConnectorCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.ForkConnectorView

```
package kiel.editor.graph;

/**
 * Describes a ForkConnector in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.14 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class ForkConnectorView extends PseudoStateView {

    /**
     * @param cell .
     */
    public ForkConnectorView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.GraphicalObjectCell

```
package kiel.editor.graph;

import org.jgraph.graph.DefaultGraphCell;

/**
 * Describes a GraphicalObject in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.10 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class GraphicalObjectCell extends DefaultGraphCell {

    /**
     * @param userObject .
     */
    public GraphicalObjectCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.HistoryCell

```
package kiel.editor.graph;

/**
 * Describes a History in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class HistoryCell extends PseudoStateCell {

    /**
     * @param userObject .
     */
    public HistoryCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.HistoryRenderer

```
package kiel.editor.graph;

/**
 * Describes a History in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.7 $ last modified $Date: 2005/03/15 11:52:53 $
 */
class HistoryRenderer extends PseudoStateRenderer {

    /**
     * @see kiel.editor.graph.PseudoStateRenderer#getMyName()
     */
    protected String getMyName() {
        return "H";
//          image = ResourceLoader.get("HistoryGrey.gif").getImage();
    }
}
```

kiel.editor.graph.HistoryView

```
package kiel.editor.graph;

/**
 * Describes a History in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.17 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class HistoryView extends PseudoStateView {

    /**
     * @param cell .
     */
    public HistoryView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.InitialArcCell

```java
package kiel.editor.graph;

/**
 * Describes an InitialArc in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class InitialArcCell extends TransitionCell {

    /**
     *
     */
    public InitialArcCell() {
        super();
    }

    /**
     * @param userObject .
     */
    public InitialArcCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.InitialArcView

```java
package kiel.editor.graph;

/**
 * Describes an InitialArc in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.14 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class InitialArcView extends TransitionView {

    /**
     * @param cell .
     */
    public InitialArcView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.InitialStateCell

```
package kiel.editor.graph;

/**
 * Describes an InitialState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class InitialStateCell extends PseudoStateCell {

    /**
     * @param userObject .
     */
    public InitialStateCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.InitialStateRenderer

```java
package kiel.editor.graph;

/**
 * Describes an InitialState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.7 $ last modified $Date: 2005/03/15 11:52:54 $
 */
class InitialStateRenderer extends PseudoStateRenderer {

    /**
     * @see kiel.editor.graph.PseudoStateRenderer#getMyName()
     */
    protected String getMyName() {
        return "I";
//          image = ResourceLoader.get("InitialStateGrey.gif").getImage();
    }
}
```

kiel.editor.graph.InitialStateView

```
package kiel.editor.graph;

/**
 * Describes an InitialState in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.17 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class InitialStateView extends PseudoStateView {

    /**
     * @param cell .
     */
    public InitialStateView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.JoinCell

```
package kiel.editor.graph;

/**
 * Describes a Join in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class JoinCell extends PseudoStateCell {

    /**
     * @param userObject .
     */
    public JoinCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.JoinView

```
package kiel.editor.graph;

/**
 * Describes a Join in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.14 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class JoinView extends PseudoStateView {

    /**
     * @param cell .
     */
    public JoinView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.JunctionCell

```
package kiel.editor.graph;

/**
 * Describes a Junction in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class JunctionCell extends PseudoStateCell {

    /**
     * @param userObject .
     */
    public JunctionCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.JunctionView

```
package kiel.editor.graph;

/**
 * Describes a Junction in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.14 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class JunctionView extends PseudoStateView {

    /**
     * @param cell .
     */
    public JunctionView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.MyGraphConstants

```java
package kiel.editor.graph;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Point;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import java.util.Map;

import javax.swing.BorderFactory;
import javax.swing.SwingConstants;

import kiel.dataStructure.ANDState;
import kiel.dataStructure.CompositeState;
import kiel.dataStructure.GraphicalObject;
import kiel.dataStructure.Node;
import kiel.dataStructure.NormalTermination;
import kiel.dataStructure.ORState;
import kiel.dataStructure.Region;
import kiel.dataStructure.StrongAbortion;
import kiel.dataStructure.Transition;
import kiel.editor.MyGraphModelUtilities;
import kiel.graphicalInformations.CompositeStateLayoutInformation;
import kiel.graphicalInformations.NodeLayoutInformation;
import kiel.graphicalInformations.View;

import org.jgraph.graph.AttributeMap;
import org.jgraph.graph.GraphConstants;

/**
 * Provides access to the JGraph cells' attributes.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.47 $ last modified $Date: 2005/07/25 11:44:37 $
 */
public class MyGraphConstants extends GraphConstants {

    /**
     * Represents a Strong abort arrow begin decoration.
     */
    public static final int ARROW_BEGIN = 100;

    /**
     * Comment for <code>GRAPHICAL_OBJECT</code>.
     */
    private static final String GRAPHICAL_OBJECT = "GRAPHICAL_OBJECT";

    /**
     * Comment for <code>EXPANDED_BOUNDS</code>.
     */
```

```java
private static final String EXPANDED_BOUNDS = "EXPANDED_BOUNDS";

/**
 * .
 */
private static final String IS_COLLAPSED = "IS_COLLAPSED";

/**
 *
 * @param map .
 * @param b .
 */
public static final void setCollapsed(final Map map, final boolean b) {
    map.put(IS_COLLAPSED, new Boolean(b));
}

/**
 *
 * @param map .
 * @return .
 */
public static boolean isCollapsed(final Map map) {
    return map.get(IS_COLLAPSED) != null
        && ((Boolean) map.get(IS_COLLAPSED)).booleanValue();
}

/**
 * @param map .
 * @return .
 */
public static final Rectangle2D getExpandedBounds(final AttributeMap map
 ) {
    return (Rectangle2D) map.get(EXPANDED_BOUNDS);
}

/**
 * @param map .
 * @param expandedBounds .
 */
public static final void setExpandedBounds(final AttributeMap map,
    final Rectangle2D expandedBounds) {
    map.put(EXPANDED_BOUNDS, expandedBounds);
}

/**
 * .
 * @param map .
 * @param graphicalObject .
 */
public static final void setGraphicalObject(
    final AttributeMap map, final GraphicalObject graphicalObject) {
```

```java
        map.put(GRAPHICAL_OBJECT, graphicalObject);
    }
    /**
     * .
     * @param map .
     * @param transition .
     */
    public static final void setTransitionAttributes(final AttributeMap map,
        final Transition transition) {

        setGraphicalObject(map, transition);
        setValue(map, transition.getLabel().toString());
        setLineEnd(map, GraphConstants.ARROW_TECHNICAL);
        setLineStyle(map, GraphConstants.STYLE_SPLINE);
        setLabelAlongEdge(map, false);
        final int pos = 20;
        setExtraLabelPositions(map, new Point[]{new Point(pos, pos)});
        GraphConstants.setExtraLabels(map,
            new Object[]{"<" + transition.getPriority().getValue() + ">"});
        if (transition instanceof StrongAbortion
            || transition instanceof NormalTermination) {
            setLineBegin(map, MyGraphConstants.ARROW_BEGIN);
        }
    }

    /**
     * .
     * @param map .
     * @param node .
     * @param kielView .
     */
    public static final void setNodeAttributes(final AttributeMap map,
        final Node node, final View kielView) {
        setGraphicalObject(map, node);
        if (node instanceof ORState || node instanceof ANDState) {
            CompositeStateLayoutInformation layout =
                (CompositeStateLayoutInformation)
            kielView.getLayoutInformation((CompositeState) node);
            setCollapsed(map, layout.isCollapsed());
        } else if (node instanceof Region) {
            // TODO Check if that is ok (dependent on EStudio-parser)
            setCollapsed(map, false);
        }
        NodeLayoutInformation layout = kielView.getLayoutInformation(node);
        Point2D offset = MyGraphModelUtilities.calculateOffset(node,
     kielView);
        if (layout != null) {
            if (isCollapsed(map)) {
                Dimension d = MyGraphModelUtilities.getInitialDimensionOf(
                    ORState.class);
                setBounds(map, new AttributeMap.SerializableRectangle2D(
                    layout.getXPos() + offset.getX(),
                    layout.getYPos() + offset.getY(),
                    d.getWidth(),
                    d.getHeight()));
                setExpandedBounds(map, new AttributeMap.
     SerializableRectangle2D(
                    layout.getXPos() + offset.getX(),
                    layout.getYPos() + offset.getY(),
                    layout.getWidth(),
                    layout.getHeight()));
            } else {
                setBounds(map, new AttributeMap.SerializableRectangle2D(
                        layout.getXPos() + offset.getX(),
                        layout.getYPos() + offset.getY(),
                        layout.getWidth(),
                        layout.getHeight()));
            }
        }
        setVerticalAlignment(map, SwingConstants.TOP);
        setBorderColor(map, Color.black);
        setBorder(map, BorderFactory.createEmptyBorder());
        setBackground(map, Color.white);
        setOpaque(map, false);
    }

    /**
     * @param map .
     * @return .
     */
    public static final synchronized GraphicalObject getGraphicalObject(
        final Map map) {
        return (GraphicalObject) map.get(GRAPHICAL_OBJECT);
    }
}
```

kiel.editor.graph.NodeCell

```
package kiel.editor.graph;

/**
 * Describes a Node in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class NodeCell extends GraphicalObjectCell {

    /**
     * @param userObject .
     */
    public NodeCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.NodeRenderer

```java
package kiel.editor.graph;

import java.awt.Graphics;

import kiel.editor.MyJGraph;
import kiel.graphicalInformations.Properties;

import org.jgraph.graph.VertexRenderer;
import org.jgraph.graph.VertexView;

/**
 * Describes a Node in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/04/01 12:26:33 $
 */
class NodeRenderer extends VertexRenderer {

    /**
     * .
     * @return .
     */
    protected MyJGraph getGraph() {
        return (MyJGraph) graph;
    }

    /**
     * .
     * @return .
     */
    protected VertexView getView() {
        return view;
    }

    /**
     *
     * @see java.awt.Component#paint(java.awt.Graphics)
     */
    public void paint(final Graphics g) {
        setFont(Properties.getDescriptionFont());
        setForeground(((MyJGraph) graph).getViewsBlack(view));
        super.paint(g);
    }
}
```

kiel.editor.graph.NodeView

```java
package kiel.editor.graph;

import java.awt.geom.Rectangle2D;

import org.jgraph.graph.CellHandle;
import org.jgraph.graph.CellViewRenderer;
import org.jgraph.graph.GraphConstants;
import org.jgraph.graph.GraphContext;
import org.jgraph.graph.VertexView;

/**
 * Describes a Node in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.33 $ last modified $Date: 2005/07/25 11:44:37 $
 */
public class NodeView extends VertexView implements Previewable {

    /**
     *
     * @see org.jgraph.graph.AbstractCellView#getRenderer()
     */
    public CellViewRenderer getRenderer() {
        return CellViewRendererFactory.get(getClass());
    }

    /**
     * @see org.jgraph.graph.CellView#isLeaf()
     */
    public final boolean isLeaf() {
        return true;
    }

    /**
     * @param cell .
     */
    public NodeView(final Object cell) {
        super(cell);
    }

    /**
     * Comment for <code>isResizing</code>.
     */
    private boolean isResizing;

    /**
     * .
     * @param aBounds .
     */
    public final void setPreviewBounds(final Rectangle2D aBounds) {
        previewBounds = aBounds;
    }

    /**
     * .
     * @param b .
     */
    public final void setResizing(final boolean b) {
        isResizing = b;
    }

    /**
     * @return .
     */
    public final boolean isResizing() {
        return isResizing;
    }

    /**
     * @see org.jgraph.graph.CellView#getHandle(org.jgraph.graph.
     GraphContext)
     */
    public final CellHandle getHandle(final GraphContext context) {
        if (GraphConstants.isSizeable(getAllAttributes())
                && context.getGraph().isSizeable()) {

            return new MySizeHandle(this, context);
        }
        return null;
    }

    /**
     *
     */
    private Rectangle2D previewBounds;

    /**
     * .
     * @return .
     */
    public final Rectangle2D getPreviewBounds() {
        return previewBounds;
    }
}
```

kiel.editor.graph.NormalTerminationCell

```java
package kiel.editor.graph;

/**
 * Describes a NormalTermination in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class NormalTerminationCell extends TransitionCell {

    /**
     *
     */
    public NormalTerminationCell() {
        super();
    }

    /**
     * @param userObject .
     */
    public NormalTerminationCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.NormalTerminationRenderer

```java
package kiel.editor.graph;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Shape;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Point2D;

import kiel.editor.resources.Preferences;

/**
 * Describes a NormalTermination in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/06/13 07:00:19 $
 */
class NormalTerminationRenderer extends TransitionRenderer {

    /**
     * @see java.awt.Component#paint(java.awt.Graphics)
     */
    public void paint(final Graphics g) {
        super.paint(g);
        g.setColor(getGraph().getViewsGreen());
        if (getView().beginShape != null) { // TODO Why can be null??
            try {
                ((Graphics2D) g).fill(getView().beginShape);
                g.setColor(getGraph().getViewsBlack(getView()));
                ((Graphics2D) g).draw(getView().beginShape);
            } catch (NullPointerException e) {
                getLogFile().log(1, "at sun.java2d.pipe.DuctusShapeRenderer
."
                    + "renderPath(Unknown Source)");
                getLogFile().log(1, "at sun.java2d.pipe.DuctusShapeRenderer
."
                    + "draw(Unknown Source)");
            }
        }
    }

    /**
     * @see org.jgraph.graph.EdgeRenderer#createLineEnd(int, int,
     * java.awt.geom.Point2D, java.awt.geom.Point2D)
     */
    protected Shape createLineEnd(final int size, final int style,
        final Point2D src, final Point2D dst) {
        if (style == MyGraphConstants.ARROW_BEGIN) {
            return new Ellipse2D.Double(dst.getX(), dst.getY(),
                Preferences.getLineEndDiameter(),
                Preferences.getLineEndDiameter());
        } else {
            return super.createLineEnd(size, style, src, dst);
        }
    }
}
```

kiel.editor.graph.NormalTerminationView

```
package kiel.editor.graph;

/**
 * Describes a NormalTermination in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.19 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class NormalTerminationView extends TransitionView {

    /**
     * @param cell .
     */
    public NormalTerminationView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.ORStateCell

```
package kiel.editor.graph;

/**
 * Describes a ORState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class ORStateCell extends CompositeStateCell {

    /**
     * @param userObject .
     */
    public ORStateCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.ORStateRenderer

```java
package kiel.editor.graph;

import java.awt.Graphics;

import kiel.graphicalInformations.Properties;

/**
 * Describes a ORState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.9 $ last modified $Date: 2005/03/15 11:52:53 $
 */
class OrStateRenderer extends CompositeStateRenderer {

    /**
     * @see org.jgraph.graph.VertexRenderer#paintSelectionBorder(
     * java.awt.Graphics)
     */
    protected void paintSelectionBorder(final Graphics g) {
        super.paintSelectionBorder(g);
        g.drawLine(0, Properties.getCompositeStateHeaderHeight(),
            getSize().width - 2,
            Properties.getCompositeStateHeaderHeight());
    }
}
```

kiel.editor.graph.ORStateView

```java
package kiel.editor.graph;

/**
 * Describes an ORState in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.18 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class ORStateView extends CompositeStateView {

    /**
     * @param cell .
     */
    public ORStateView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.PseudoStateCell

```
package kiel.editor.graph;

/**
 * Describes a PseudoState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class PseudoStateCell extends NodeCell {

    /**
     * @param userObject .
     */
    public PseudoStateCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.PseudoStateRenderer

```java
package kiel.editor.graph;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;

import kiel.editor.MyJGraph;
import kiel.editor.resources.Preferences;

import org.jgraph.graph.VertexRenderer;
import org.jgraph.graph.VertexView;

/**
 * Describes a PseudoState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.11 $ last modified $Date: 2005/04/01 16:24:06 $
 */
class PseudoStateRenderer extends VertexRenderer {

    /**
     * .
     * @return .
     */
    protected MyJGraph getGraph() {
        return (MyJGraph) graph;
    }

    /**
     * .
     * @return .
     */
    protected VertexView getView() {
        return view;
    }

    /**
     *
     * @return .
     */
    protected String getMyName() {
        return ""; // H, H*, I, S: wird im Kreis angezeigt
    }

    /**
     *
     * @return .
     */
    protected Color getMyColor() {
        return ((MyJGraph) graph).getViewsGray();
    }

//      protected Image image = ResourceLoader.get("attention.gif").getImage
    ();

    /**
     * @param g .
     */
    protected void paintSelectionBorder(final Graphics g) {
        ((Graphics2D) g).setStroke(getGraph().getStroke(getView()));
        Dimension d = getSize(); g.setColor(
            ((MyJGraph) graph).getViewsBlack(view));
        g.drawOval(0, 0, d.width - 1, d.height - 1);
        g.setColor(getMyColor());
        g.fillOval(0, 0, d.width - 1, d.height - 1);
        g.setColor(((MyJGraph) graph).getViewsBlack(view));
        g.drawString(getMyName(),
            Preferences.getPseudoStateNameOffset().x,
            Preferences.getPseudoStateNameOffset().y);

//          g.drawImage(((PseudoStateView) view).image, 0, 0, null);
    }
}
```

kiel.editor.graph.PseudoStateView

```
package kiel.editor.graph;

/**
 * Describes a PseudoState in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.23 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class PseudoStateView extends NodeView {

    /**
     * @param cell .
     */
    public PseudoStateView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.RegionCell

```java
package kiel.editor.graph;

/**
 * Describes a Region in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class RegionCell extends CompositeStateCell {

    /**
     * @param userObject .
     */
    public RegionCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.RegionRenderer

```java
package kiel.editor.graph;

import java.awt.Graphics;

/**
 * Describes a Region in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.8 $ last modified $Date: 2005/03/21 22:53:58 $
 */
class RegionRenderer extends CompositeStateRenderer {

    /**
     * @see org.jgraph.graph.VertexRenderer#paintSelectionBorder(
     * java.awt.Graphics)
     */
    protected void paintSelectionBorder(final Graphics g) {
        final int initialYOffset = -10;
        drawAllLabels(g, initialYOffset);
    }
}
```

kiel.editor.graph.RegionView

```
package kiel.editor.graph;

/**
 * Describes a Region in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.17 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class RegionView extends CompositeStateView {

    /**
     * @param cell .
     */
    public RegionView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.SimpleStateCell

```java
package kiel.editor.graph;

/**
 * Describes a SimpleState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class SimpleStateCell extends StateCell {

    /**
     * @param userObject .
     */
    public SimpleStateCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.SimpleStateView

```java
package kiel.editor.graph;

/**
 * Describes a SimpleState in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.15 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class SimpleStateView extends StateView {

    /**
     * @param cell .
     */
    public SimpleStateView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.StateCell

```java
package kiel.editor.graph;

/**
 * Describes a State in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class StateCell extends NodeCell {

    /**
     * @param userObject .
     */
    public StateCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.StateRenderer

```java
package kiel.editor.graph;

import java.awt.Component;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.util.Arrays;
import java.util.List;

import kiel.dataStructure.State;
import kiel.editor.resources.Preferences;
import kiel.editor.resources.ResourceBundle;
import kiel.graphicalInformations.Properties;

import org.jgraph.JGraph;
import org.jgraph.graph.CellView;
import org.jgraph.graph.DefaultGraphCell;

/**
 * Describes a State in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/07/14 15:27:56 $
 */
class StateRenderer extends NodeRenderer {

    /**
     * @see org.jgraph.graph.CellViewRenderer#getRendererComponent(
     * org.jgraph.JGraph, org.jgraph.graph.CellView, boolean, boolean,
     boolean)
     */
    public Component getRendererComponent(final JGraph graph,
        final CellView view, final boolean sel, final boolean focus,
        final boolean preview) {

        super.getRendererComponent(graph, view, sel, focus, preview);
        if (!(view instanceof CompositeStateView)) {
            setText("");
        }
        return this;
    }

    /**
     * @see org.jgraph.graph.VertexRenderer#paintSelectionBorder(
     * java.awt.Graphics)
     */
    protected void paintSelectionBorder(final Graphics g) {
        ((Graphics2D) g).setStroke(getGraph().getStroke(getView()));
        g.setColor(getGraph().getViewsBlack(getView()));
        g.drawRoundRect(0, 0, getSize().width - 1,
            getSize().height - 1,
                Preferences.getArcWidthHeight(),
                Preferences.getArcWidthHeight());
            drawAllLabels(g, Properties.getHeight(
                Properties.getActivityFont(), "X") + HEADER_FOOTER);

            if (!(view instanceof CompositeStateView)) {
                int offset = Properties.getUpperOffset((State)
                    MyGraphConstants.getGraphicalObject(
                        ((NodeCell) view.getCell()).getAttributes()));
                String s = graph.convertValueToString(view);
                g.setFont(Properties.getDescriptionFont());
                g.drawString(
                    s,
                    (getWidth() - Properties.getWidth(g.getFont(), s)) / 2,
                    2 + 2 + 2 + offset + (getHeight() - offset) / 2);
            }
        }

        /**
         *
         * @param g .
         * @param initialYOffset .
         */
        protected void drawAllLabels(final Graphics g, final int initialYOffset)
          {
            int yOffset = initialYOffset;
            yOffset = drawVariablesAndEvents(g, yOffset);
            drawActions(g, yOffset);
        }

        /**
         * .
         * @param g .
         * @param yOffset .
         * @return .
         */
        protected int drawVariablesAndEvents(final Graphics g, final int yOffset
         ) {
            return yOffset;
        }

        /**
         * .
         * @param rowTitle .
         * @param elements .
         * @param aYOffset .
         * @param g .
         * @return .
         */
        protected int draw(final String rowTitle, final List elements,
            final int aYOffset, final Graphics g) {
```

```java

        int yOffset = aYOffset;

        if (elements != null) {
            String elementsString = "";
            for (int i = 0; i < elements.size(); i++) {
                if (elements.get(i) != null) {
                    if (i > 0) {
                        elementsString += ", ";
                    }
                    elementsString += elements.get(i);
                }
            }
            if (elementsString.length() > 0) {
                final int xOffset = 10;
                g.setFont(Properties.getActivityFont());
                g.drawString(rowTitle + elementsString, xOffset, yOffset);
                g.drawLine(0,
                    yOffset + HEADER_FOOTER,
                    getSize().width - 2,
                    yOffset + HEADER_FOOTER);
                yOffset += Properties.getHeight(
                    Properties.getActivityFont(), "X") + 2 * HEADER_FOOTER;
            }
        }
        return yOffset;
    }

    /**
     * .
     */
    private static final int HEADER_FOOTER = 3;

    /**
     * .
     * @param g .
     * @param initialYOffset .
     */
    protected void drawActions(final Graphics g, final int initialYOffset) {
        int yOffset = initialYOffset;
        if (Preferences.getShowStateActions()) {
            State state = (State) MyGraphConstants.getGraphicalObject(
                ((DefaultGraphCell) getView().getCell()).getAttributes());
            yOffset = draw(ResourceBundle.getString("On_Entry_Action") + ":
 ",
                Arrays.asList(state.getEntry().getActions()), yOffset, g);
            yOffset = draw(ResourceBundle.getString("On_Inside_Action") + ":
 ",
                Arrays.asList(state.getDoActivity().getActions()), yOffset,
 g);
            yOffset = draw(ResourceBundle.getString("On_Exit_Action") + ":
 ",
                Arrays.asList(state.getExit().getActions()), yOffset, g);
        }
    }
}
```

kiel.editor.graph.StateView

```
package kiel.editor.graph;

/**
 * Describes a State in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.19 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class StateView extends NodeView {

    /**
     * @param cell .
     */
    public StateView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.StrongAbortionCell

```
package kiel.editor.graph;

/**
 * Describes a StrongAbortion in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class StrongAbortionCell extends TransitionCell {

    /**
     *
     */
    public StrongAbortionCell() {
        super();
    }

    /**
     * @param userObject .
     */
    public StrongAbortionCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.StrongAbortionRenderer

```java
package kiel.editor.graph;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Shape;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Point2D;

import kiel.editor.resources.Preferences;

/**
 * Describes a StrongAbortion in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/06/13 07:00:19 $
 */
class StrongAbortionRenderer extends TransitionRenderer {

    /**
     * @see java.awt.Component#paint(java.awt.Graphics)
     */
   public void paint(final Graphics g) {
        super.paint(g);
        g.setColor(getGraph().getViewsRed());
        if (getView().beginShape != null) { // TODO Why can be null??
            try {
                ((Graphics2D) g).fill(getView().beginShape);
                g.setColor(getGraph().getViewsBlack(getView()));
                ((Graphics2D) g).draw(getView().beginShape);
            } catch (NullPointerException e) {
                getLogFile().log(1, "at sun.java2d.pipe.DuctusShapeRenderer
 ."
                    + "renderPath(Unknown Source)");
                getLogFile().log(1, "at sun.java2d.pipe.DuctusShapeRenderer
 ."
                    + "draw(Unknown Source)");
            }
        }
    }

    /**
     * @see org.jgraph.graph.EdgeRenderer#createLineEnd(int, int,
     * java.awt.geom.Point2D, java.awt.geom.Point2D)
     */
    protected Shape createLineEnd(final int size, final int style,
        final Point2D src, final Point2D dst) {
        if (style == MyGraphConstants.ARROW_BEGIN) {
            return new Ellipse2D.Double(dst.getX(), dst.getY(),
                Preferences.getLineEndDiameter(),
                Preferences.getLineEndDiameter());
        } else {
            return super.createLineEnd(size, style, src, dst);
        }
    }
}
```

kiel.editor.graph.StrongAbortionView

```
package kiel.editor.graph;

/**
 * Describes a StrongAbortion in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.20 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class StrongAbortionView extends TransitionView {

    /**
     * @param cell .
     */
    public StrongAbortionView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.SuspendCell

```
package kiel.editor.graph;

/**
 * Describes a Suspend in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class SuspendCell extends PseudoStateCell {

    /**
     * @param userObject .
     */
    public SuspendCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.SuspendRenderer

```
package kiel.editor.graph;

import java.awt.Color;

/**
 * Describes a Suspend in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.8 $ last modified $Date: 2005/04/01 12:33:58 $
 */
class SuspendRenderer extends PseudoStateRenderer {

    /**
     * @see kiel.editor.graph.PseudoStateRenderer#getMyName()
     */
    protected String getMyName() {
        return "S";
//          image = ResourceLoader.get("Suspend.gif").getImage();
    }

    /**
     * @see kiel.editor.graph.PseudoStateRenderer#getMyColor()
     */
    protected Color getMyColor() {
        return getGraph().getViewsYellow();
    }
}
```

kiel.editor.graph.SuspendView

```
package kiel.editor.graph;

/**
 * Describes a Suspend in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.18 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class SuspendView extends PseudoStateView {

    /**
     * @param cell .
     */
    public SuspendView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.SuspensionCell

```
package kiel.editor.graph;

/**
 * Describes a Suspension in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class SuspensionCell extends TransitionCell {

    /**
     *
     */
    public SuspensionCell() {
        super();
    }

    /**
     * @param userObject .
     */
    public SuspensionCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.SuspensionRenderer

```
package kiel.editor.graph;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Shape;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Point2D;

import kiel.editor.resources.Preferences;

/**
 * Describes a Suspension in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.3 $ last modified $Date: 2005/06/13 07:00:18 $
 */
class SuspensionRenderer extends TransitionRenderer {

    /**
     * @see java.awt.Component#paint(java.awt.Graphics)
     */
    public void paint(final Graphics g) {
        super.paint(g);
        if (getView().endShape != null) { // TODO Why can be null??
            try {
                g.setColor(getGraph().getViewsBlack(getView()));
                ((Graphics2D) g).draw(getView().endShape);
                g.setColor(getGraph().getViewsYellow());
                ((Graphics2D) g).fill(getView().endShape);
            } catch (NullPointerException e) {
                getLogFile().log(1, "at sun.java2d.pipe.DuctusShapeRenderer
."
                    + "renderPath(Unknown Source)");
                getLogFile().log(1, "at sun.java2d.pipe.DuctusShapeRenderer
."
                    + "draw(Unknown Source)");
            }
        }
    }

    /**
     * @see org.jgraph.graph.EdgeRenderer#createLineEnd(int, int,
     * java.awt.geom.Point2D, java.awt.geom.Point2D)
     */
    protected Shape createLineEnd(final int size, final int style,
        final Point2D src, final Point2D dst) {
        Shape shape = super.createLineEnd(size, style, src, dst);
        if (style != MyGraphConstants.ARROW_BEGIN) {
            return new Ellipse2D.Double(dst.getX(), dst.getY(),
                Preferences.getLineEndDiameter(),
                Preferences.getLineEndDiameter());
        } else {
            return shape;
        }
    }
}
```

kiel.editor.graph.SuspensionView

```java
package kiel.editor.graph;

/**
 * Describes a Suspension in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.15 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class SuspensionView extends TransitionView {

    /**
     * @param cell .
     */
    public SuspensionView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.SynchStateCell

```
package kiel.editor.graph;

/**
 * Describes a SynchState in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.12 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class SynchStateCell extends PseudoStateCell {

    /**
     * @param userObject .
     */
    public SynchStateCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.SynchStateView

```
package kiel.editor.graph;

/**
 * Describes a SynchState in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.15 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class SynchStateView extends PseudoStateView {

    /**
     * @param cell .
     */
    public SynchStateView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.TransitionCell

```java
package kiel.editor.graph;

import java.awt.geom.Point2D;
import java.util.ArrayList;
import java.util.List;

import org.jgraph.graph.AttributeMap;
import org.jgraph.graph.GraphConstants;

/**
 * Describes a Transition in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.19 $ last modified $Date: 2005/07/25 11:44:37 $
 */
public class TransitionCell extends EdgeCell {

    /**
     * Comment for <code>DUMMY_POINT</code>.
     */
    public static final Point2D DUMMY_POINT =
        new AttributeMap.SerializablePoint2D(
            Double.NEGATIVE_INFINITY, Double.NEGATIVE_INFINITY);

    /**
     * @see org.jgraph.graph.DefaultEdge#checkDefaults()
     */
    protected final void checkDefaults() {
        List points = GraphConstants.getPoints(attributes);
        if (points == null) {
            List defaultPoints = new ArrayList();
            defaultPoints.add(DUMMY_POINT);
            defaultPoints.add(DUMMY_POINT);
            GraphConstants.setPoints(attributes, defaultPoints);
        }
        super.checkDefaults();
    }

    /**
     *
     * @return .
     */
    public final boolean hasOnlyDummyPoints() {
        List points = GraphConstants.getPoints(attributes);
        return points != null
            && points.size() == 2
            && DUMMY_POINT.equals(points.get(0))
            && DUMMY_POINT.equals(points.get(1));
    }

    /**
     *
     */
    public TransitionCell() {
        super();
    }

    /**
     * @param userObject .
     */
    public TransitionCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.TransitionRenderer

```
package kiel.editor.graph;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Shape;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;

import kiel.editor.MyJGraph;
import kiel.editor.resources.Preferences;
import kiel.graphicalInformations.Properties;
import kiel.util.LogFile;

import org.jgraph.graph.EdgeRenderer;
import org.jgraph.graph.EdgeView;

/**
 * Describes a Transition in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.25 $ last modified $Date: 2005/07/27 09:42:14 $
 */
class TransitionRenderer extends EdgeRenderer {

    /**
     * Workaround (JGraph throws a NullPointerException).
     * @see org.jgraph.graph.EdgeRenderer#getExtraLabelBounds(
     * org.jgraph.graph.EdgeView, int)
     */
    public Rectangle2D getExtraLabelBounds(
        final EdgeView aView, final int index) {
        if (graph == null) {
            return null;
        } else {
            return super.getExtraLabelBounds(aView, index);
        }
    }

    /**
     * Workaround (JGraph throws a NullPointerException).
     * @see org.jgraph.graph.EdgeRenderer#getExtraLabelSize(
     * org.jgraph.graph.EdgeView, int)
     */
    public Dimension getExtraLabelSize(final EdgeView aView, final int index
     ) {
        if (graph == null) {
            return null;
        } else {
            return super.getExtraLabelSize(aView, index);
        }
    }

    /**
     * The logging handle. Prints logs to standard out.
     */
    private LogFile logFile = new LogFile("Editor", 2);

    /**
     *
     * @return .
     */
    protected LogFile getLogFile() {
        return logFile;
    }

    /**
     * .
     * @return .
     */
    protected MyJGraph getGraph() {
        return (MyJGraph) graph;
    }

    /**
     * .
     * @return .
     */
    protected EdgeView getView() {
        return view;
    }

    /**
     * Workaround for setting the label color.
     * @see org.jgraph.graph.EdgeRenderer#paintLabel(
     * java.awt.Graphics, java.lang.String, java.awt.geom.Point2D, boolean)
     */
    protected void paintLabel(final Graphics g, final String label,
        final Point2D p, final boolean mainLabel) {
        fontColor = getGraph().getViewsBlack(view);
        super.paintLabel(g, label, p, mainLabel);
    }

    /**
     * @see java.awt.Component#paint(java.awt.Graphics)
     */
    public void paint(final Graphics g) {
        setForeground(getGraph().getViewsBlack(view));
        setFont(Properties.getLabelFont());
        super.paint(g);
        if (getView().endShape != null) {
```

```
            ((Graphics2D) g).fill(getView().endShape);
        }
    }

    /**
     * @see org.jgraph.graph.EdgeRenderer#getLabelPosition(
     * org.jgraph.graph.EdgeView)
     */
    public Point2D getLabelPosition(final EdgeView aView) {
        Point2D p = super.getLabelPosition(aView);
        p.setLocation(p.getX() + Preferences.getLabelOffset().x,
            p.getY() + Preferences.getLabelOffset().y);
        return p;
    }

    /**
     * TODO why? This method is needed even if TransitionRenderers'
     * subclasses do not have any code and even if TransitionRenderer itself
     * doen't have any code as well.
     * @see org.jgraph.graph.EdgeRenderer#createShape()
     */
    protected Shape createShape() {
        final int times = 10;
        for (int i = 1; i <= times; i++) {
            try {
                Shape result = super.createShape();
                if (result != null) {
                    return result;
                } else {
                    logFile.log(1, "org.jgraph.graph.EdgeRenderer.
 createShape()"
                        + " returned null.");
                }
            } catch (NullPointerException e) {
                logFile.log(1, "EdgeRenderer.createShape() threw"
                    + " NullpointerException.");
            }
        }
        return null;
    }
}
```

kiel.editor.graph.TransitionView

```
package kiel.editor.graph;

/**
 * Describes a Transition in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.25 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class TransitionView extends EdgeView {

    /**
     * @param cell .
     */
    public TransitionView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.graph.WeakAbortionCell

```
package kiel.editor.graph;

/**
 * Describes a WeakAbortion in the JGraph model.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/03/15 11:52:53 $
 */
public class WeakAbortionCell extends TransitionCell {

    /**
     *
     */
    public WeakAbortionCell() {
        super();
    }

    /**
     * @param userObject .
     */
    public WeakAbortionCell(final Object userObject) {
        super(userObject);
    }
}
```

kiel.editor.graph.WeakAbortionView

```java
package kiel.editor.graph;

/**
 * Describes a WeakAbortion in the JGraph view.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.15 $ last modified $Date: 2005/03/15 11:52:54 $
 */
public class WeakAbortionView extends TransitionView {

    /**
     * @param cell .
     */
    public WeakAbortionView(final Object cell) {
        super(cell);
    }
}
```

kiel.editor.resources.Preferences

```java
package kiel.editor.resources;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Point;
import java.awt.Rectangle;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Properties;
import java.util.Set;
import java.util.StringTokenizer;

/**
 * Handles the user preferences, that can be edited through the
 * File-->Preferences menu item.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.36 $ last modified $Date: 2005/08/01 17:46:35 $
 */
public final class Preferences {

    /**
     * .
     *
     */
    private Preferences() {

    }

    /**
     * components can register to be notified whenever the preferences have
     * changed, that is, when the user edited the preferences table in the
     * preferences dialog.
     * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
     a>
     */
    public interface Listener {

        /**
         * Is called whenever the preferences values have changed.
         */
        void preferencesChanged();
    }

    /**
     * The user editable preferences that are loaded and stored in the user
     home
     * directory.
     */
    private static final Properties PREFERENCES = new Properties();

    /**
     * The default and non editable preferences, that are stored in this
     * package. These values are needed whenever the user types in permitted
     * values for some preferences.
     */
    private static final Properties DEFAULT_PREFERENCES = new Properties();

    /**
     * Some preferences are not just Strings or ints, but are quite complex
     * objects, like fonts or rectangles, so once they are requested, they
     are
     * build and stored. They will be replaced from this cache, whenever
     they
     * change (through setProperty().
     */
    private static HashMap cache = new HashMap();

    /**
     * The listeners to be notified whenever the preferences changed.
     */
    private static ArrayList listeners = new ArrayList();

    /**
     * the directory where the preferences are stored.
     */
    private static final String WORKING_DIR =
        System.getProperty("user.home")
        + File.separator
        + ".kiel"
        + File.separator;

    /**
     * The preferences file name without directory.
     */
    private static final String FILENAME = "MyPreferences.properties";

    /**
     * loads all the preferences.
     */
    static {
        initialize();
    }

    /**
     *
```

```
     *
     */
    private static void initialize() {
        try {
            DEFAULT_PREFERENCES.load(
                Preferences.class.getResourceAsStream(FILENAME));
            if (new File(WORKING_DIR + FILENAME).exists()) {
                PREFERENCES.load(new FileInputStream(WORKING_DIR + FILENAME)
     );
            } else {
                PREFERENCES.load(
                    Preferences.class.getResourceAsStream(FILENAME));
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     *
     *
     */
    public static void setDefaultValues() {
        File f = new File(WORKING_DIR + FILENAME);
        if (f.exists()) {
            f.delete();
        }
        initialize();
        notifyListeners();
    }

    /**
     * Add a listener to the listeners' list.
     * @param listener .
     */
    public static void addListener(final Listener listener) {
        listeners.add(listener);
    }

    /**
     * Stores the current PREFERENCES hashmap.
     */
    public static void store() {
        try {
            PREFERENCES.store(new FileOutputStream(
                WORKING_DIR + FILENAME), null);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Returns all the preferences keys.
     * @return .
     */
    public static Set keySet() {
        return PREFERENCES.keySet();
    }

    /**
     * Returns a particular preference value.
     * @param key .
     * @return .
     */
    public static String getProperty(final String key) {
        return PREFERENCES.getProperty(key);
    }

    /**
     * Sets the value for a particular preference.
     * @param key .
     * @param value .
     */
    public static void setProperty(final String key, final String value) {
        cache.remove(key);
        PREFERENCES.setProperty(key, value);
        notifyListeners();
    }

    /**
     *
     *
     */
    private static void notifyListeners() {
        for (int i = 0; i < listeners.size(); i++) {
            ((Listener) listeners.get(i)).preferencesChanged();
        }
    }

    /**
     * Returns the preference value of the given key as int.
     * If that fails (because of an invalid value in the preferences file),
     the
     * appropriate value of the default preferences is returned.
     * @param key .
     * @return .
     */
    private static int getInt(final String key) {
        Integer result = (Integer) cache.get(key);
        if (result == null) {
            try {
                result = new Integer(PREFERENCES.getProperty(key));
            } catch (NumberFormatException e) {
                setProperty(key, DEFAULT_PREFERENCES.getProperty(key));
                return getInt(key);
            }
            cache.put(key, result);
        }
```

```java
        return result.intValue();
    }

    /**
     * Returns the preference value of the given key as double.
     * If that fails (because of an invalid value in the preferences file),
     the
     * appropriate value of the default preferences is returned.
     * @param key .
     * @return .
     */
    private static double getDouble(final String key) {
        Double result = (Double) cache.get(key);
        if (result == null) {
            try {
                result = new Double(PREFERENCES.getProperty(key));
            } catch (NumberFormatException e) {
                setProperty(key, DEFAULT_PREFERENCES.getProperty(key));
                return getDouble(key);
            }
            cache.put(key, result);
        }
        return result.doubleValue();
    }

    /**
     * Returns the preference value of the given key as boolean.
     * If that fails (because of an invalid value in the preferences file),
     the
     * appropriate value of the default preferences is returned.
     * @param key .
     * @return .
     */
    private static boolean getBoolean(final String key) {
        Boolean result = (Boolean) cache.get(key);
        if (result == null) {
            try {
                result = new Boolean(PREFERENCES.getProperty(key));
            } catch (NumberFormatException e) {
                setProperty(key, DEFAULT_PREFERENCES.getProperty(key));
                return getBoolean(key);
            }
            cache.put(key, result);
        }
        return result.booleanValue();
    }

    /**
     * Returns the preference value of the given key as point.
     * If that fails (because of an invalid value in the preferences file),
     the
     * appropriate value of the default preferences is returned.
     * @param key .
     * @return .
     */
    private static Point getPoint(final String key) {
        Point result = (Point) cache.get(key);
        if (result == null) {
            try {
                StringTokenizer st =
                    new StringTokenizer(PREFERENCES.getProperty(key), ",");
                result = new Point(
                    Integer.parseInt(st.nextToken()),
                    Integer.parseInt(st.nextToken()));
            } catch (NumberFormatException e) {
                setProperty(key, DEFAULT_PREFERENCES.getProperty(key));
                return getPoint(key);
            }
            cache.put(key, result);
        }
        return result;
    }

    /**
     * Returns the preference value of the given key as dimension.
     * If that fails (because of an invalid value in the preferences file),
     the
     * appropriate value of the default preferences is returned.
     * @param key .
     * @return .
     */
    private static Dimension getDimension(final String key) {
        Dimension result = (Dimension) cache.get(key);
        if (result == null) {
            try {
                StringTokenizer st =
                    new StringTokenizer(PREFERENCES.getProperty(key), ",");
                result = new Dimension(
                    Integer.parseInt(st.nextToken()),
                    Integer.parseInt(st.nextToken()));
            } catch (NumberFormatException e) {
                setProperty(key, DEFAULT_PREFERENCES.getProperty(key));
                return getDimension(key);
            }
            cache.put(key, result);
        }
        return result;
    }

    /**
     * Returns the preference value of the given key as rectangle.
     * If that fails (because of an invalid value in the preferences file),
     the
     * appropriate value of the default preferences is returned.
     * @param key .
     * @return .
     */
    private static Rectangle getRectangle(final String key) {
```

```
        Rectangle result = (Rectangle) cache.get(key);
        if (result == null) {
            try {
                StringTokenizer st =
                    new StringTokenizer(PREFERENCES.getProperty(key), ",");
                result = new Rectangle(
                    Integer.parseInt(st.nextToken()),
                    Integer.parseInt(st.nextToken()),
                    Integer.parseInt(st.nextToken()),
                    Integer.parseInt(st.nextToken()));
            } catch (NumberFormatException e) {
                setProperty(key, DEFAULT_PREFERENCES.getProperty(key));
                return getRectangle(key);
            }
            cache.put(key, result);
        }
        return result;
    }

    /**
     * Returns the preference value of the given key as color.
     * If that fails (because of an invalid value in the preferences file),
     the
     * appropriate value of the default preferences is returned.
     * @param key .
     * @return .
     */
    private static Color getColor(final String key) {
        Color result = (Color) cache.get(key);
        if (result == null) {
            try {
                StringTokenizer st =
                    new StringTokenizer(PREFERENCES.getProperty(key), ",");
                result = new Color(
                    Integer.parseInt(st.nextToken()),
                    Integer.parseInt(st.nextToken()),
                    Integer.parseInt(st.nextToken()));
            } catch (NumberFormatException e) {
                setProperty(key, DEFAULT_PREFERENCES.getProperty(key));
                return getColor(key);
            }
            cache.put(key, result);
        }
        return result;
    }

    /**
     * Returns the GuiColorLight.
     * @return .
     */
    public static Color getGuiColorLight() {
        return getColor("guiColorLight");
    }

    /**
     * Returns the guiColorDark.
     * @return .
     */
    public static Color getGuiColorDark() {
        return getColor("guiColorDark");
    }

    /**
     * Returns the portColor.
     * @return .
     */
    public static Color getPortColor() {
        return getColor("portColor");
    }

    /**
     * Returns the guiButtonColorSelectedLight.
     * @return .
     */
    public static Color getGuiButtonColorSelectedLight() {
        return getColor("guiButtonColorSelectedLight");
    }

    /**
     * Returns the guiButtonColorSelectedDark.
     * @return .
     */
    public static Color getGuiButtonColorSelectedDark() {
        return getColor("guiButtonColorSelectedDark");
    }

    /**
     * Returns the intersectionSensitivity.
     * @return .
     */
    public static int getIntersectionSensitivity() {
        return getInt("intersectionSensitivity");
    }

    /**
     * Returns the arcWidthHeight.
     * @return .
     */
    public static int getArcWidthHeight() {
        return getInt("arcWidthHeight");
    }

    /**
     * Returns the compositeStateImageOffset.
     * @return .
     */
    public static int getCompositeStateImageOffset() {
        return getInt("compositeStateImageOffset");
```

```
    }

    /**
     * Returns the finalStateBorder.
     * @return .
     */
    public static int getFinalStateBorder() {
        return getInt("finalStateBorder");
    }

    /**
     * Returns the labelOffset.
     * @return .
     */
    public static Point getLabelOffset() {
        return getPoint("labelOffset");
    }

    /**
     * Returns the lineEndDiameter.
     * @return .
     */
    public static int getLineEndDiameter() {
        return getInt("lineEndDiameter");
    }

    /**
     * Returns the pseudoStateNameOffset.
     * @return .
     */
    public static Point getPseudoStateNameOffset() {
        return getPoint("pseudoStateNameOffset");
    }

    /**
     * Returns the zoomInFactor.
     * @return .
     */
    public static double getZoomInFactor() {
        return getDouble("zoomInFactor");
    }

    /**
     * Returns the zoomOutFactor.
     * @return .
     */
    public static double getZoomOutFactor() {
        return getDouble("zoomOutFactor");
    }

    /**
     * Returns the replayStepTime.
     * @return .
     */
    public static int getReplayStepTime() {
        return getInt("replayStepTime");
    }

    /**
     * Returns the replayStepTimeFast.
     * @return .
     */
    public static int getReplayStepTimeFast() {
        return getInt("replayStepTimeFast");
    }

    /**
     * Returns the flipArea.
     * @return .
     */
    public static Rectangle getFlipArea() {
        return getRectangle("flipArea");
    }

    /**
     * Returns the gridSize.
     * @return .
     */
    public static int getGridSize() {
        return getInt("gridSize");
    }

    /**
     * Returns the cellSelectionTolerance.
     * @return .
     */
    public static int getCellSelectionTolerance() {
        return getInt("cellSelectionTolerance");
    }

    /**
     * Returns the gridEnabled.
     * @return .
     */
    public static boolean getGridEnabled() {
        return getBoolean("gridEnabled");
    }

    /**
     * Returns the gridVisible.
     * @return .
     */
    public static boolean getGridVisible() {
        return getBoolean("gridVisible");
    }

    /**
     * Returns the overviewColor.
```

```java
 * @return .
 */
public static Color getOverviewColor() {
    return getColor("overviewColor");
}

/**
 * Returns the lastFilesCount.
 * @return .
 */
public static int getLastFilesCount() {
    return getInt("lastFilesCount");
}

/**
 * Returns the language.
 * @return .
 */
public static String getLanguage() {
    return PREFERENCES.getProperty("language");
}

/**
 * Returns the layoutOnInsert.
 * @return .
 */
public static boolean getLayoutOnInsert() {
    return getBoolean("layoutOnInsert");
}

/**
 * Returns the showSplashScreen.
 * @return .
 */
public static boolean getShowSplashScreen() {
    return getBoolean("showSplashScreen");
}

/**
 * Returns the showOverviewLayer.
 * @return .
 */
public static boolean getShowOverviewLayer() {
    return getBoolean("showOverviewLayer");
}

/**
 * Returns the gridVisible.
 * @return .
 */
public static boolean getShowGrid() {
    return getBoolean("gridVisible");
}

/**
 *
 *
 */
public static void toggleShowOverviewLayer() {
    setProperty("showOverviewLayer",
        (!getBoolean("showOverviewLayer")) + "");
}

/**
 *
 *
 */
public static void toggleShowGrid() {
    setProperty("gridVisible",
        (!getBoolean("gridVisible")) + "");
}

/**
 * Returns the showReplayToolBarOnLoad.
 * @return .
 */
public static boolean getShowReplayToolBarOnLoad() {
    return getBoolean("showReplayToolBarOnLoad");
}

/**
 * Returns the showAlignmentToolBarOnLoad.
 * @return .
 */
public static boolean getShowAlignmentToolBarOnLoad() {
    return getBoolean("showAlignmentToolBarOnLoad");
}

/**
 * Returns the showStateActions.
 * @return .
 */
public static boolean getShowStateActions() {
    return getBoolean("showStateActions");
}

/**
 * Returns the showStateActions.
 * @return .
 */
public static boolean getShowCollapseButtons() {
    return getBoolean("showCollapseButtons");
}

/**
 * .
 *
 */
```

```java
public static void toggleShowStateActions() {
    setProperty("showStateActions", (!getShowStateActions()) + "");
}

/**
 * Returns the showStateActions.
 * @return .
 */
public static boolean getShowCompositeStateVariables() {
    return getBoolean("showCompositeStateVariables");
}

/**
 * .
 *
 */
public static void toggleShowCompositeStateVariables() {
    setProperty("showCompositeStateVariables",
        (!getShowCompositeStateVariables()) + "");
}

/**
 * Returns the showStateActions.
 * @return .
 */
public static boolean getShowCompositeStateEvents() {
    return getBoolean("showCompositeStateEvents");
}

/**
 * .
 *
 */
public static void toggleShowCompositeStateEvents() {
    setProperty("showCompositeStateEvents",
        (!getShowCompositeStateEvents()) + "");
}

/**
 * Returns the morphingLayouterSpeed.
 * @return .
 */
public static int getMorphingLayouterSpeed() {
    return getInt("morphingLayouterSpeed");
}

/**
 * Returns the morphingLayouterBorderDistance.
 * @return .
 */
public static int getMorphingLayouterBorderDistance() {
    return getInt("morphingLayouterBorderDistance");
}

/**
 * Returns the morphingLayouterCheckInterval.
 * @return .
 */
public static int getMorphingLayouterCheckInterval() {
    return getInt("morphingLayouterCheckInterval");
}

/**
 * Returns the defaultState.
 * @return .
 */
public static Dimension getDefaultState() {
    return getDimension("defaultState");
}

/**
 * Returns the defaultNode.
 * @return .
 */
public static Dimension getDefaultNode() {
    return getDimension("defaultNode");
}

/**
 * Returns the defaultRootState.
 * @return .
 */
public static Dimension getDefaultRootState() {
    return getDimension("defaultRootState");
}

/**
 * Returns the placeOfDeepHistory.
 * @return .
 */
public static Point getPlaceOfDeepHistory() {
    return getPoint("placeOfDeepHistory");
}

/**
 * Returns the initialGraphOffset.
 * @return .
 */
public static int getInitialGraphOffset() {
    return getInt("initialGraphOffset");
}

/**
 * Returns the validAreaBorder.
 * @return .
 */
public static int getValidAreaBorder() {
    return getInt("validAreaBorder");
```

```java
    }

    /**
     * Returns the traceLogSeparator.
     * @return .
     */
    public static String getTraceLogSeparator() {
        return PREFERENCES.getProperty("traceLogSeparator");
    }

    /**
     * Returns the sizeHandleSize.
     * @return .
     */
    public static int getSizeHandleSize() {
        return getInt("sizeHandleSize");
    }

    /**
     * Returns the sizeHandleSensitivity.
     * @return .
     */
    public static int getSizeHandleSensitivity() {
        return getInt("sizeHandleSensitivity");
    }

    /**
     * Returns the sizeHandleColor.
     * @return .
     */
    public static Color getSizeHandleColor() {
        return getColor("sizeHandleColor");
    }

    /**
     * Returns the gridVisible.
     * @return .
     */
    public static boolean getSizeHandleVisible() {
        return getBoolean("sizeHandleVisible");
    }

    /**
     * Returns the sizeHandleSensitivity.
     * @return .
     */
    public static int getMorphingLayouterStepInterval() {
        return getInt("morphingLayouterStepInterval");
    }
}
```

kiel.editor.resources.ResourceBundle

```java
package kiel.editor.resources;

import java.util.Locale;
import java.util.MissingResourceException;

import kiel.util.LogFile;

/**
 * Provides access to the user language dependent displayed strings, like
     label
 * texts, menu texts, ...
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.25 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ResourceBundle implements Preferences.Listener {

    /**
     * The singleton instance.
     */
    private static ResourceBundle instance = new ResourceBundle();

    /**
     * The logging handle. Prints logs to standard out.
     */
    private static LogFile logFile = new LogFile("Editor", 2);

    /**
     * Returns the value of the current resource bundle for the given key.
     * @param key .
     * @return .
     */
    public static String getString(final String key) {
        if (instance.myResources != null) {
            try {
                String value = instance.myResources.getString(key);
                if (value != null) {
                    return value;
                }
            } catch (MissingResourceException e) {
                logFile.log(2, "Key " + key + " not found in ResourceBundle
 .");
                return key;
            }
        }
        return key;
    }

    /**
     * Reference to the language dependent current resource bundle file.
     */
    private java.util.ResourceBundle myResources;

    /**
     * Is called when the user preferences have changed. If the user
     preferenced
     * language has changed, then another resource bundle has to be loaded.
     * @see kiel.editor.resources.Preferences.Listener#preferencesChanged()
     */
    public void preferencesChanged() {
        System.setProperty("user.language", Preferences.getLanguage());
        myResources = java.util.ResourceBundle.getBundle(
            "kiel.editor.resources.MyResources",
            new Locale(Preferences.getLanguage()));
    }

    /**
     * Creates the ResourceBundle object and adds itself to the Preferences
     * listener list.
     */
    private ResourceBundle() {
        Preferences.addListener(this);
        preferencesChanged();
    }
}
```

kiel.editor.resources.ResourceLoader

```java
package kiel.editor.resources;

import java.util.Hashtable;

import javax.swing.ImageIcon;

/**
 * Loads resources.
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.19 $ last modified $Date: 2005/07/25 11:44:36 $
 */
public final class ResourceLoader {

    /**
     * Caches the already loaded images.
     */
    private static Hashtable images = new Hashtable();

    /**
     * Returns the image.
     * @param filename .
     * @return .
     */
    public static ImageIcon get(final String filename) {
        if (filename == null) {
            return null;
        }
        Object result = images.get(filename);
        if (result == null) {
            result = new ImageIcon(ResourceLoader.class.getResource(filename
 ));
            images.put(filename, result);
        }
        return (ImageIcon) result;
    }

    /**
     * .
     *
     */
    private ResourceLoader() {
    }
}
```

kiel.util.main.IKielFrame

```java
package kiel.util.main;

import java.io.File;
import java.util.Collection;

import javax.swing.JFrame;
import javax.swing.JMenu;

import kiel.dataStructure.StateChart;
import kiel.graphicalInformations.View;

/**
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.13 $ last modified $Date: 2005/07/14 15:31:09 $
 */
public interface IKielFrame {

    /**
     * @return .
     */
    Object getFileInterfaceModel();

    /**
     * @param file .
     */
    void setCurrentFile(File file);

    /**
     * For use as parent for dialogs
     * @return .
     */
    JFrame getJFrame();

    /**
     * @return .
     */
    File getCurrentFile();

    /**
     * @param file .
     */
    void open(File file);

    /**
     * @param displayMessage .
     * @param isError .
     * @param e .
     */
    void handleException(final String displayMessage, final boolean isError,
        final Exception e);

    /**
     *
     */
    void resizeToolbarPanel();

    /**
     *
     * @param layouterName .
     * @return .
     */
    KielLayouter getKielLayouterByName(String layouterName);

    /**
     * Calls kiel.fileInterface.esterelstudio.GraphParser.parseSignals(
     String)
     * @param text .
     * @return .
     * @throws Exception an EstudioParseException
     */
    Collection parseSignals(String text) throws Exception;

    /**
     * Calls kiel.fileInterface.esterelstudio.NodeParser.parseVariables(
     String)
     * @param text .
     * @return .
     * @throws Exception an EstudioParseException
     */
    Collection parseVariables(String text) throws Exception;

    /**
     * .
     * @return .
     */
    File getFileInterfaceSaveDir();

    /**
     * Calls FileInterfaceModel.setStateChart(StateChart);
     * Calls FileInterfaceModel.setActualView(View);
     * Calls FileInterfaceModel.writeStateChartFile(
     *     getCurrentFile(), FileInterfaceModel.getLastFileFilter());
     * @param stateChart .
     * @param view .
     * @throws Exception .
     */
    void writeStateChartFile(StateChart stateChart, View view)
        throws Exception;

    /**
```

```
 * Calls FileInterfaceModel.setStateChart(StateChart);
 * Calls FileInterfaceModel.setActualView(View);
 * Calls FileInterfaceModel.showSaveFileDialog();
 * @param stateChart .
 * @param view .
 * @throws Exception .
 */
void writeStateChartFileWithSaveDialog(StateChart stateChart, View view)
    throws Exception;

/**
 * Calls FileInterfaceModel.getStatechartFile();
 * @return .
 */
File getStatechartFile();

/**
 * Dynamically creates the Layouter menu depending on what
 * Handler.getHandeler().getLayouterNames() will return. Hopefully
 * more then none:-)
 * @param isRootMenu is this menu shown in the menu bar?
 * @param actionClass the Action class which contains code to be
 executed
 * when a layout request is handled. Browser and Editor have to provide
 * their own action class with a constructor taking the name of the
 * layouter.
 * @return a new layouter menu object.
 */
JMenu createLayouterMenu(boolean isRootMenu, Class actionClass);

/**
 * .
 * @param question .
 * @return .
 */
boolean ask(String question);
}
```

kiel.util.main.KielComponent

```
package kiel.util.main;

import javax.swing.ImageIcon;
import javax.swing.JComponent;
import javax.swing.JMenu;
import javax.swing.JMenuItem;
import javax.swing.JTabbedPane;
import javax.swing.JToolBar;

import kiel.dataStructure.StateChart;
import kiel.graphicalInformations.View;

/**
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.8 $ last modified $Date: 2005/03/12 02:09:59 $
 */
public interface KielComponent {

    /**
     * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
     a>
     */
    public final class ToolBar {

        /**
         *
         */
        private JToolBar jToolBar;

        /**
         *
         */
        private String menuName;

        /**
         *
         */
        private boolean isVisible;

        /**
         * @param aJToolBar .
         * @param aMenuName .
         * @param anIsVisible .
         */
        public ToolBar(final JToolBar aJToolBar, final String aMenuName,
            final boolean anIsVisible) {
            jToolBar = aJToolBar;
            menuName = aMenuName;
            isVisible = anIsVisible;
        }

        /**
         * @return .
         */
        public JToolBar getJToolBar() {
            return jToolBar;
        }

        /**
         * @param toolBar .
         */
        public void setJToolBar(final JToolBar toolBar) {
            this.jToolBar = toolBar;
        }

        /**
         * @return .
         */
        public String getMenuName() {
            return menuName;
        }

        /**
         * @return .
         */
        public boolean isVisible() {
            return isVisible;
        }

        /**
         * @param anIsVisible .
         */
        public void setVisible(final boolean anIsVisible) {
            isVisible = anIsVisible;
        }
    }

    /**
     * @return all menus that this component has, except for the file menu
     * and the help menu.
     */
    JMenu[] getRootMenus();

    /**
     * @return all leading menu items in the file menu. they are followed by
     * the file open menu item.
     */
    JMenuItem[] getFileMenuItems1();

    /**
```

```
     * @return all menu items in the file menu between the print and the
     exit
     * menu item
     */
    JMenuItem[] getFileMenuItems2();

    /**
     * @return .
     */
    ToolBar[] getToolBars();

    /**
     * @return the components' main panel, e.g. the browser display panel or
     * the editor graph panel.
     */
    JComponent getMainComponent();

    /**
     * @return the tree view. can be null.
     */
    JComponent getTreePanel();

    /**
     * @return sub panels. e.g., console output, simulation output. can be
     null.
     */
    JTabbedPane getSubPanels();

    /**
     * Sets the parent frame of the component. this frame should be used as
     * parent for dialogs.
     * @param parentFrame the main application frame.
     */
    void setKielFrame(IKielFrame parentFrame);

    /**
     * the main application frame components should be able to exchange
     their
     * current displayed / used statechart.
     * @param statechart the current statechart
     * @param view the current view
     */
    void setCurrentStateChart(StateChart statechart, View view);

    /**
     * the main application frame components should be able to exchange
     their
     * current displayed / used statechart.
     * @return statechart the current statechart
     */
    StateChart getCurrentStateChart();

    /**
     * the main application frame components should be able to exchange
     their
     * current displayed / used statechart.
     * @return view the current view
     */
    View getCurrentView();

    /**
     * @return an image icon for the switch button
     */
    ImageIcon getImageIcon();

    /**
     * @return the component in the right corner of the status line.
     */
    JComponent getStatusLineComponent();

    /**
     * @return .
     */
    String getKielFrameTitle();
}
```

kiel.util.main.KielLayouter

```java
package kiel.util.main;

import kiel.dataStructure.StateChart;
import kiel.graphicalInformations.View;
import kiel.configMngr.Configuration;

/**
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.3 $ last modified $Date: 2005/07/18 15:04:19 $
 */
public interface KielLayouter {

    /**
     *
     * @param statechart .
     * @throws Exception .
     */
    void setStatechart(StateChart statechart) throws Exception;

    /**
     *
     * @return .
     */
    View getStaticView();

    /**
     *
     * @param view .
     */
    void layoutView(View view);

    /**
     * @param config Configuration
     * @return View
     */
    View getConfigurationView(Configuration config);

    /**
     * Set the static view for the pseudolayouter.
     * @param v View
     */
    void setStaticView(View v);
}
```

kiel.KielFrame

```java
package kiel;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.net.URL;
import java.util.Collection;
import java.util.Properties;

import javax.jnlp.BasicService;
import javax.jnlp.ServiceManager;
import javax.swing.AbstractAction;
import javax.swing.AbstractButton;
import javax.swing.Action;
import javax.swing.ImageIcon;
import javax.swing.JCheckBox;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
import javax.swing.JSplitPane;
import javax.swing.JTabbedPane;
import javax.swing.JToolBar;
import javax.swing.JWindow;
import javax.swing.KeyStroke;
import javax.swing.SwingUtilities;
import javax.swing.UIManager;
import javax.swing.border.LineBorder;
import javax.swing.plaf.ColorUIResource;
import javax.swing.plaf.metal.DefaultMetalTheme;
import javax.swing.plaf.metal.MetalLookAndFeel;

import kiel.browser.Browser;
import kiel.dataStructure.StateChart;
import kiel.editor.Editor;
import kiel.editor.FileHistory;
import kiel.editor.Utils;
import kiel.editor.resources.Preferences;
import kiel.editor.resources.ResourceBundle;
import kiel.editor.resources.ResourceLoader;
import kiel.fileInterface.FileInterfaceException;
import kiel.fileInterface.FileInterfaceModel;
import kiel.fileInterface.FileInterfaceProperties;
import kiel.fileInterface.esterelstudio.GraphParser;
import kiel.fileInterface.esterelstudio.NodeParser;
import kiel.graphicalInformations.View;
import kiel.layouter.Handler;
import kiel.layouter.LayoutException;
import kiel.layouter.Layouter;
import kiel.layouter.PseudoLayouter;
import kiel.optimizer.OptimizerChooser;
import kiel.util.MatlabProcessInfo;
import kiel.util.main.IKielFrame;
import kiel.util.main.KielComponent;
import kiel.util.main.KielLayouter;
import kiel.configMngr.Configuration;

/**
 * <p>Copyright: (c) 2004 <a href="http://luepke.com">Florian Luepke</a></p>
 * <p>Company: Uni Kiel</p>.
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </a>
 * @version $Revision: 1.84 $ last modified $Date: 2005/08/01 17:55:23 $
 */
public final class KielFrame extends JFrame implements IKielFrame,
    Preferences.Listener {

    /**
     * @param args .
     * @throws Exception .
     */
    public static void main(final String[] args) throws Exception {

        String jwsHome = System.getProperty("jnlpx.deployment.user.home");

        if (jwsHome != null) {  // Windows environment
            jwsHome += "/javaws/cache";
        } else { // Unix environment
            jwsHome = System.getProperty("deployment.javaws.splash.index");
            if (jwsHome != null && jwsHome.indexOf("/cache/javaws") != -1) {
                jwsHome = jwsHome.substring(
                    0, jwsHome.indexOf("/cache/javaws"));
                jwsHome += "/cache/javaws";
```

```java
        }
    }
    if (jwsHome != null && jwsHome.trim().length() > 0) {
        System.setProperty("java.library.path",
            jwsHome + System.getProperty("layouter.path"));
        System.setProperty("layouter.path.windows",
            System.getProperty("java.library.path"));
        System.setProperty("layouter.path.other",
            System.getProperty("java.library.path"));
        FileInterfaceProperties.getInstance().setProperty(
            FileInterfaceProperties.PLUGINDIR,
            jwsHome + System.getProperty("fileInterface.path"));
    }
    setPreferencesDependentUIDefaults();
    UIManager.getDefaults().put("Menu.background",
        new ColorUIResource(Color.WHITE));
    UIManager.getDefaults().put("MenuItem.background",
        new ColorUIResource(Color.WHITE));
    UIManager.getDefaults().put("CheckBoxMenuItem.background",
        new ColorUIResource(Color.WHITE));
    UIManager.getDefaults().put("Separator.background",
        new ColorUIResource(Color.WHITE));
    UIManager.getDefaults().put("Separator.shadow",
        new ColorUIResource(Color.WHITE));
    UIManager.getDefaults().put("Separator.highlight",
        new ColorUIResource(Color.WHITE));
    UIManager.getDefaults().put("Separator.foreground",
        new ColorUIResource(Color.WHITE));

    MetalLookAndFeel.setCurrentTheme(new DefaultMetalTheme() {

        /**
         * @see javax.swing.plaf.metal.MetalTheme#getSecondary1()
         */
        protected ColorUIResource getSecondary1() {
            return new ColorUIResource(Preferences.getGuiColorDark());
        }

        /**
         * @see javax.swing.plaf.metal.MetalTheme#getSecondary2()
         */
        protected ColorUIResource getSecondary2() {
            return new ColorUIResource(Preferences.getGuiColorDark());
        }

        /**
         * @see javax.swing.plaf.metal.MetalTheme#getSecondary3()
         */
        protected ColorUIResource getSecondary3() {
            return new ColorUIResource(Preferences.getGuiColorDark());
        }

        /**
         * @see javax.swing.plaf.metal.MetalTheme#getControlInfo()
         */
        public ColorUIResource getControlInfo() {
            return new ColorUIResource(Preferences.getGuiColorDark());
        }

        /**
         * @see javax.swing.plaf.metal.MetalTheme#getControl()
         */
        public ColorUIResource getControl() {
            return new ColorUIResource(Preferences.getGuiColorLight());
        }

        /**
         * @see javax.swing.plaf.metal.MetalTheme#getMenuBackground()
         */
        public ColorUIResource getMenuBackground() {
            return new ColorUIResource(Preferences.getGuiColorLight());
        }
    });

    JWindow w = new JWindow();
    Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
    final int screenWidth = 341;
    final int screenHeight = 360;
    w.setBounds((d.width - screenWidth) / 2,
        (d.height - screenHeight) / 2, screenWidth, screenHeight);
    w.getContentPane().setLayout(new BorderLayout());
    w.getContentPane().add(
        new JLabel(ResourceLoader.get("splashscreen3.gif")));
    if (Preferences.getShowSplashScreen()) {
        w.setVisible(true);
    }
    KielFrame kielFrame = new KielFrame();
    kielFrame.setKielComponents(new KielComponent[]{
        new Editor(kielFrame),
        new Browser(kielFrame)});
    kielFrame.setVisible(true);
    if (Preferences.getShowSplashScreen()) {
        w.dispose();
    }
    statusLeft.setIcon(ResourceLoader.get("logo_cau.gif"));
}

/**
 *
 */
private static void setPreferencesDependentUIDefaults() {
    UIManager.getDefaults().put("SplitPaneDivider.border",
        new ColorUIResource(Preferences.getGuiColorDark()));
    UIManager.getDefaults().put("SplitPane.background",
        new ColorUIResource(Preferences.getGuiColorDark()));
    UIManager.getDefaults().put("TabbedPane.selected",
        new ColorUIResource(Preferences.getGuiColorDark()));
    UIManager.getDefaults().put("TabbedPane.tabAreaBackground",
```

```java
            new ColorUIResource(Preferences.getGuiColorLight()));
        UIManager.getDefaults().put("TabbedPane.background",
            new ColorUIResource(Preferences.getGuiColorLight()));
        UIManager.getDefaults().put("ScrollBar.foreground",
            new ColorUIResource(Preferences.getGuiColorLight()));
        UIManager.getDefaults().put("ScrollBar.background",
            new ColorUIResource(Preferences.getGuiColorLight()));
        UIManager.getDefaults().put("ScrollBar.thumbHighlight",
            new ColorUIResource(Preferences.getGuiColorDark()));
        UIManager.getDefaults().put("ScrollBar.thumbShadow",
            new ColorUIResource(Preferences.getGuiColorDark()));
        UIManager.getDefaults().put("ScrollBar.thumb",
            new ColorUIResource(Preferences.getGuiColorLight()));
        UIManager.getDefaults().put("ScrollBar.darkShadow",
            new ColorUIResource(Preferences.getGuiColorDark()));
        UIManager.getDefaults().put("ScrollBar.highlight",
            new ColorUIResource(Preferences.getGuiColorLight()));
        UIManager.getDefaults().put("ScrollBar.shadow",
            new ColorUIResource(Preferences.getGuiColorLight()));
        UIManager.getDefaults().put("Slider.highlight",
            new ColorUIResource(Preferences.getGuiColorLight()));
        UIManager.getDefaults().put("Slider.darkShadow",
            new ColorUIResource(Preferences.getGuiColorDark()));
        UIManager.getDefaults().put("Slider.thumb",
            new ColorUIResource(Preferences.getGuiColorLight()));
        UIManager.getDefaults().put("Menu.selectionBackground",
            new ColorUIResource(Preferences.getGuiColorDark()));
        UIManager.getDefaults().put("MenuItem.selectionBackground",
            new ColorUIResource(Preferences.getGuiColorDark()));
    }

    /**
     * @see kiel.util.main.IKielFrame#createLayouterMenu(
     *  boolean, java.lang.Class)
     */
    public JMenu createLayouterMenu(final boolean isRootMenu,
        final Class actionClass) {

        JMenu layout;
        if (isRootMenu) {
            layout = Utils.createJMenu("ActionLayout", "layoutMnemonic");
        } else {
            layout = Utils.createJMenu("ActionLayout", "layoutMnemonic",
                "layout.gif");
        }

        String[] layouters = (String[]) Handler.instance().getLayouterNames
 ()
            .toArray(new String[0]);

        for (int i = 0; i < layouters.length; i++) {
            try {
                Action action = (Action) actionClass.getConstructor(
                    new Class[]{String.class}).newInstance(
                        new Object[]{layouters[i]});
                JMenuItem mi = new JMenuItem(action);
                final int ctrlAndShift = 3;
                mi.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_F1 + i,
                    ctrlAndShift));
                layout.add(mi);
            } catch (Exception e) {
                handleException(e.getMessage(), true, e);
            }
        }
        return layout;
    }

    /**
     * @see kiel.util.main.IKielFrame#parseSignals(java.lang.String)
     */
    public Collection parseSignals(final String text) throws Exception {
        return GraphParser.parseSignals(text);
    }

    /**
     * @see kiel.util.main.IKielFrame#parseVariables(java.lang.String)
     */
    public Collection parseVariables(final String text) throws Exception {
        return NodeParser.parseVariables(text);
    }

    /**
     * .
     * @return .
     */
    public File getFileInterfaceSaveDir() {
        return ((FileInterfaceModel) getFileInterfaceModel()).getSaveDir();
    }

    /**
     * @see kiel.util.main.IKielFrame#writeStateChartFile(
     *  kiel.dataStructure.StateChart, kiel.graphicalInformations.View)
     */
    public void writeStateChartFile(final StateChart stateChart,
        final View view) throws Exception {

        FileInterfaceModel fileInterfaceModel =
            (FileInterfaceModel) getFileInterfaceModel();
        fileInterfaceModel.setStateChart(stateChart);
        fileInterfaceModel.setActualView(view);
        fileInterfaceModel.writeStateChartFile(
            getCurrentFile(), fileInterfaceModel.getLastFileFilter());
    }

    /**
     * @see kiel.util.main.IKielFrame#writeStateChartFileWithSaveDialog(
     *  kiel.dataStructure.StateChart, kiel.graphicalInformations.View)
     */
```

```java
public void writeStateChartFileWithSaveDialog(final StateChart
 stateChart,
    final View view) throws Exception {

    FileInterfaceModel fileInterfaceModel =
        (FileInterfaceModel) getFileInterfaceModel();
    fileInterfaceModel.setStateChart(stateChart);
    fileInterfaceModel.setActualView(view);
    fileInterfaceModel.showSaveFileDialog();
}

/**
 * @see kiel.util.main.IKielFrame#getStatechartFile()
 */
public File getStatechartFile() {
    FileInterfaceModel fileInterfaceModel =
        (FileInterfaceModel) getFileInterfaceModel();
    return fileInterfaceModel.getStatechartFile();
}

/**
 * @see kiel.util.main.IKielFrame#getKielLayouterByName(java.lang.String
 )
 */
public KielLayouter getKielLayouterByName(final String layouterName) {
    return new KielLayouter() {
        private Layouter layouter = Handler.instance()
            .getLayouterNamed(layouterName);
        public View getStaticView() {
            return layouter.getStaticView();
        }
        public void layoutView(final View view) {
            layouter.layoutView(view);
        }
        public View getConfigurationView(final Configuration config) {
          return layouter.getConfigurationView(config);
        }
        public void setStatechart(final StateChart statechart)
            throws Exception {
            layouter.setStatechart(statechart);
        }
        public void setStaticView(final View v) {
          if (layouter.getClass() == PseudoLayouter.class) {
            ((PseudoLayouter) layouter).setView(v);
          }
        }
    };
}

/**
 * @see kiel.editor.resources.Preferences.Listener#preferencesChanged()
 */
public void preferencesChanged() {
    setPreferencesDependentUIDefaults();
    for (int i = 0; kielComponents != null && i < kielComponents.length;
        i++) {
        kielComponents[i].preferencesChanged();
    }
    updateContent(current, current);
    repaint();
}

/**
 *
 */
private ExtendedKielComponent[] kielComponents;

/**
 *
 */
private int current;

/**
 *
 */
private FileInterfaceModel fileModel = new FileInterfaceModel();

/**
 *
 */
private File currentFile;

/**
 * Lists all messages that the user does not want to see anymore.
 */
private Properties suppressedMessages = new Properties();

/**
 * @see kiel.util.main.IKielFrame#ask(java.lang.String)
 */
public boolean ask(final String question) {
    if (suppressedMessages.containsKey(question + "YES")) {
        return true;
    } else if (suppressedMessages.containsKey(question + "NO")) {
        return false;
    } else {
        MessagePanel messagePanel = new MessagePanel(
            question, ResourceBundle.getString("rememberAnswer"));
        int answer = JOptionPane.showOptionDialog(
            this,
            messagePanel,
            ResourceBundle.getString("note"),
            JOptionPane.OK_CANCEL_OPTION,
            JOptionPane.QUESTION_MESSAGE,
            null, new Object[] {
                ResourceBundle.getString("yes"),
                ResourceBundle.getString("no")},
                ResourceBundle.getString("yes"));
```

```
            if (messagePanel.isCheckBoxSelected()) {
                if (answer == 0) {
                    handleSuppressMessage(question + "YES");
                } else {
                    handleSuppressMessage(question + "NO");
                }
            }
            return answer == 0;
        }
    }

    /**
     * @param displayMessage .
     * @param isApplicationError .
     * @param e .
     */
    public void handleException(final String displayMessage,
        final boolean isApplicationError, final Exception e) {

        if (isApplicationError && e != null) {
            e.printStackTrace();
        }

        if (displayMessage == null
            || suppressedMessages.containsKey(displayMessage)) {
            return;
        }

        MessagePanel messagePanel = new MessagePanel(
            displayMessage, ResourceBundle.getString("doNotShowAgain"));
        if (isApplicationError) {
            JOptionPane.showMessageDialog(this,
                messagePanel,
                ResourceBundle.getString("error"),
                JOptionPane.ERROR_MESSAGE);
        } else {
            JOptionPane.showMessageDialog(this,
                messagePanel,
                ResourceBundle.getString("note"),
                JOptionPane.INFORMATION_MESSAGE);
        }
        if (messagePanel.isCheckBoxSelected()) {
            handleSuppressMessage(displayMessage);
        }
    }

    /**
     * .
     * @param displayMessage .
     */
    private void handleSuppressMessage(final String displayMessage) {
        suppressedMessages.put(displayMessage, "");
        try {
            suppressedMessages.store(new FileOutputStream(
                new File(SUPPRESSED_MESSAGES_FILE_NAME)), null);
        } catch (IOException ex) {
            if (!(ex instanceof FileNotFoundException)) {
                handleException(ex.getMessage(), true, ex);
            }
        }
    }

    /**
     * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
     a>
     */
    private class MessagePanel extends JPanel {

        /**
         * .
         */
        private JCheckBox checkBox;

        /**
         * @param displayMessage .
         * @param checkBoxMessage .
         */
        MessagePanel(final String displayMessage,
            final String checkBoxMessage) {

            checkBox = new JCheckBox(checkBoxMessage);
            setLayout(new BorderLayout());
            JLabel label = new JLabel(displayMessage);
            add(label, "Center");
            add(checkBox, "South");
            final int height = 80;
            int width = Math.max(
                label.getPreferredSize().width,
                checkBox.getPreferredSize().width);
            setPreferredSize(new Dimension(width, height));
        }

        /**
         * .
         * @return .
         */
        boolean isCheckBoxSelected() {
            return checkBox.isSelected();
        }
    }

    /**
     * @see kiel.util.main.IKielFrame#getJFrame()
     */
    public JFrame getJFrame() {
        return this;
    }
```

```java
/**
 * @see kiel.util.main.IKielFrame#getFileInterfaceModel()
 */
public Object getFileInterfaceModel() {
    return fileModel;
}

/**
 *
 */
private void changeTitle() {
    String modelSource = "";
    if (kielComponents[current].getCurrentStateChart() != null) {
        modelSource = kielComponents[current].getCurrentStateChart()
            .getModelSource();
    }
    if (modelSource == null) {
        modelSource = "";
    }
    if (currentFile != null) {
        setTitle(currentFile.getName() + " - "
            + kielComponents[current].getKielFrameTitle() + " "
            + modelSource);
    } else {
        setTitle(kielComponents[current].getKielFrameTitle() + " "
            + modelSource);
    }
}

/**
 * Is called with null when a new Chart is being created or with the new
 * file name when the chart is saved.
 * @see kiel.util.main.IKielFrame#setCurrentFile(java.io.File)
 */
public void setCurrentFile(final File newFile) {
    currentFile = newFile;
    changeTitle();
    FileHistory.getInstance(this).add(newFile);
}

/**
 * @see kiel.util.main.IKielFrame#getCurrentFile()
 */
public File getCurrentFile() {
    return currentFile;
}

/**
 *
 */
private static final String SUPPRESSED_MESSAGES_FILE_NAME =
    System.getProperty("user.home")
    + File.separator
    + ".kiel"
    + File.separator
    + "KielFrame.properties";

/**
 *
 */
public KielFrame() {
    super(ResourceBundle.getString("KIEL_Statechart_Tool"));
    setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    Dimension d = Toolkit.getDefaultToolkit().getScreenSize();
    final int frameWidth = 850;
    final int frameHeight = 500;
    setBounds((d.width - frameWidth) / 2, (d.height - frameHeight) / 2,
        frameWidth, frameHeight);
    setIconImage(ResourceLoader.get("smileyTransparent.gif").getImage())
;

    addWindowListener(new WindowAdapter() {
        public void windowClosing(final WindowEvent e) {
            new ActionFileExit().actionPerformed(null);
        }
    });


    try {
        suppressedMessages.load(new FileInputStream(
            SUPPRESSED_MESSAGES_FILE_NAME));
    } catch (IOException e) {
        if (!(e instanceof FileNotFoundException)) {
            handleException(e.getMessage(), true, e);
        }
    }

    Preferences.addListener(this);
}

/**
 * @param aKielComponents .
 */
public void setKielComponents(final KielComponent[] aKielComponents) {
    if (aKielComponents == null) {
        kielComponents = new ExtendedKielComponent[0];
    } else {
        kielComponents = new ExtendedKielComponent[aKielComponents.
length];
    }

    final ImageIcon[] flipButtonIcons =
        new ImageIcon[aKielComponents.length];
    for (int i = 0; aKielComponents != null && i < aKielComponents.
length;
        i++) {
        kielComponents[i] = createExtendedKielComponent(aKielComponents[
i]);
```

```
            // This is needed to avoid multiple calls on getToolBar()
            //returning new Objects each time
            kielComponents[i].getMainComponent().setBorder(null);
            flipButtonIcons[i] = kielComponents[i].getImageIcon();
            if (flipButtonIcons[i] == null) {
                flipButtonIcons[i] =
                    ResourceLoader.get("smileyTransparent.gif");
            }
        }

        toolbarPanel.addMouseListener(new MouseAdapter() {
            public void mousePressed(final MouseEvent e) {
                if (SwingUtilities.isRightMouseButton(e)) {
                    kielComponents[current].getPopupMenu()
                        .show(toolbarPanel, e.getX(), e.getY());
                }
            }
        });

        final AbstractButton flipButton =
            Utils.createJButton(null, "tooltipToggleEditorBrowser");
        flipButton.setAction(new AbstractAction("",
            ResourceLoader.get("smileyTransparent.gif")) {
            public void actionPerformed(final ActionEvent e) {
                if (kielComponents.length != 0) {
                    int newCurrent = (current + 1) % kielComponents.length;
                    updateContent(current, newCurrent);
                    flipButton.setIcon(
                        flipButtonIcons[(current + 2) % kielComponents.
length]);
                    kielComponents[newCurrent].setCurrentStateChart(
                            kielComponents[current].getCurrentStateChart(),
                            kielComponents[current].getCurrentView());
                    current = newCurrent;
                    changeTitle();
                    repaint();
                }
            }
        });
        JToolBar flipToolbar = Utils.createJToolBar();
        flipToolbar.setFloatable(true);
        flipToolbar.add(flipButton);
        toolbarPanel.add(flipToolbar);
        final int toolbarSize = 32;
        toolbarPanel.setPreferredSize(new Dimension(toolbarSize, toolbarSize
));
        getContentPane().add(toolbarPanel, "North");
        getContentPane().add(centerPanel, "Center");
        final int dividerSize = 3;
        centerPanel.setDividerSize(dividerSize);
        centerPanel.setOrientation(JSplitPane.VERTICAL_SPLIT);

        statusPanel.add(statusLeft, BorderLayout.CENTER);
        final int statusHeight = 23;
        statusPanel.setPreferredSize(new Dimension(0, statusHeight));
        getContentPane().add(statusPanel, BorderLayout.SOUTH);
//        setStatus("   Christian Albrecht University Kiel");
        updateContent(current, current);
    }

    /**
     * @see kiel.util.main.IKielFrame#resizeToolbarPanel()
     */
    public void resizeToolbarPanel() {
        toolbarPanel.invalidate();
        int width = 0;
        for (int i = 0; i < toolbarPanel.getComponentCount(); i++) {
            width += toolbarPanel.getComponent(i).getPreferredSize().width;
        }
        final int toolbarSize = 32;
        toolbarPanel.setPreferredSize(
            new Dimension(toolbarSize,
                (int) (Math.ceil((double) width
                    / (double) getSize().width) * toolbarSize)));
        setVisible(true);
    }

    /**
     *
     */
    private JPanel statusPanel = new JPanel(new BorderLayout());

    /**
     *
     */
    private static JLabel statusLeft = Utils.createJLabel();

    /**
     * @param text .
     */
    public static void setStatus(final String text) {
        statusLeft.setText(text);
    }

    /**
     *
     */
    private JPanel toolbarPanel =
        Utils.createJPanel(new FlowLayout(FlowLayout.LEFT, 0, 0));

    /**
     *
     */
    private JSplitPane centerPanel = new JSplitPane();

    /**
     * @param currentIndex .
     * @param newIndex .
```

```
 */
private void updateContent(final int currentIndex, final int newIndex) {

    ExtendedKielComponent currentKielComponent =
        kielComponents[currentIndex];
    ExtendedKielComponent newKielComponent = kielComponents[newIndex];

    kielComponents[newIndex].setKielFrame(this);

    for (int i = 0; i < currentKielComponent.getToolBars().length; i++)
 {
        if (currentKielComponent.getToolBars()[i].isVisible()) {
            toolbarPanel.remove(
                currentKielComponent.getToolBars()[i].getJToolBar());
        }
    }
    for (int i = 0; i < newKielComponent.getToolBars().length; i++) {
        if (newKielComponent.getToolBars()[i].isVisible()) {
            toolbarPanel.add(
                newKielComponent.getToolBars()[i].getJToolBar());
        }
    }
    getContentPane().remove(currentKielComponent.getCompleteComponent())
 ;
    getContentPane().add(newKielComponent.getCompleteComponent(), "
 Center");

    resizeToolbarPanel();
    setStatusPanelVisible(newKielComponent.isStatusPanelVisible());

    if (currentKielComponent.getStatusLineComponent() != null) {
        statusPanel.remove(currentKielComponent.getStatusLineComponent()
 );
    }
    if (newKielComponent.getStatusLineComponent() != null) {
        statusPanel.add(newKielComponent.getStatusLineComponent(), "East
 ");
    }

    JMenuBar menubar = Utils.createJMenuBar();
    setJMenuBar(menubar);

    JMenu menu = Utils.createJMenu("file", "fileMnemonic");
    menubar.add(menu);

    JMenuItem[] fileMenuItems1 = null;
    if (kielComponents.length != 0) {
        fileMenuItems1 = (JMenuItem[]) newKielComponent.
 getFileMenuItems1();
    }
    if (fileMenuItems1 != null) {
        for (int i = 0; i < fileMenuItems1.length; i++) {
            menu.add(fileMenuItems1[i]);
        }
        menu.addSeparator();
    }
    JMenuItem mi = new JMenuItem(new ActionFileOpen());
    mi.setAccelerator(KeyStroke.getKeyStroke('O', 2));
    menu.add(mi);
    menu.addSeparator();

    JMenuItem[] fileMenuItems2 = null;
    if (kielComponents.length != 0) {
        fileMenuItems2 = (JMenuItem[]) newKielComponent.
 getFileMenuItems2();
    }
    if (fileMenuItems2 != null) {
        for (int i = 0; i < fileMenuItems2.length; i++) {
            menu.add(fileMenuItems2[i]);
        }
        menu.addSeparator();
    }

    menu.add(FileHistory.getInstance(this).getMenu());
    menu.addSeparator();
    mi = new JMenuItem(new ActionResetMessages());
//      mi.setAccelerator(KeyStroke.getKeyStroke('Q', 2));
    menu.add(mi);
    menu.addSeparator();
    mi = new JMenuItem(new ActionFileExit());
    mi.setAccelerator(KeyStroke.getKeyStroke('Q', 2));
    menu.add(mi);

    JMenu[] rootMenus = null;
    if (kielComponents.length != 0) {
        rootMenus = (JMenu[]) newKielComponent.getRootMenus();
    }
    if (rootMenus != null) {
        for (int i = 0; i < rootMenus.length; i++) {
            menubar.add(rootMenus[i]);
        }
    }

    menu = Utils.createJMenu("optimize", "optimizeMnemonic");
    menubar.add(menu);
    menu.add(new ActionOptimizeAll());
    menu.add(new ActionOptimizeLevel());

    menu = Utils.createJMenu("help", "helpMnemonic");
    menubar.add(menu);
    menu.add(new ActionHelpAbout());
    menu.add(new ActionHelpEditor());
    setVisible(true);
}

/**
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
 a>
```

```
 */
abstract class KielFrameAction extends AbstractAction
    implements Preferences.Listener {

    /**
     * Adds the created object to the preferences listener list.
     * @param iconName .
     */
    protected KielFrameAction(final String iconName) {
        super(null, ResourceLoader.get(iconName));
        Preferences.addListener(this);
        preferencesChanged();
    }

    /**
     * Changes the Action.NAME according to the changes in the
 Preferences
     * class.
     * @see kiel.editor.resources.Preferences.Listener#
 preferencesChanged()
     */
    public final void preferencesChanged() {
        putValue(Action.NAME, ResourceBundle.getString(getClass().
 getName()
            .substring(getClass().getName().lastIndexOf('$') + 1)));
    }
}
/**
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
 a>
 */
class ActionHelpAbout extends KielFrameAction {

    /**
     *
     */
    ActionHelpAbout() {
        super("Help16.gif");
        setEnabled(false);
    }

    /**
     * @see java.awt.event.ActionListener#
     * actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(final ActionEvent e) {
    }
}

/**
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
 a>
 */
class ActionHelpEditor extends KielFrameAction {

    /**
     *
     */
    ActionHelpEditor() {
        super("Edit16.gif");
    }

    /**
     * @see java.awt.event.ActionListener#
     * actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(final ActionEvent e) {
        try {
            ((BasicService) ServiceManager
                .lookup("javax.jnlp.BasicService")).showDocument(
                    new URL(System.getProperty("editor.manual.path")));
        } catch (Exception ex) {
            handleException(ResourceBundle.getString("jnlpUnavailable"),
                false, ex);
        }
    }
}

/**
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
 a>
 */
class ActionOptimizeAll extends KielFrameAction {

    /**
     *
     */
    ActionOptimizeAll() {
        super("empty.gif");
    }

    /**
     * @see java.awt.event.ActionListener#
     * actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(final ActionEvent e) {
        optimize(true);
    }
}

/**
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
 a>
 */
class ActionOptimizeLevel extends KielFrameAction {

    /**
     *
```

```java
     */
    ActionOptimizeLevel() {
        super("empty.gif");
    }

    /**
     * @see java.awt.event.ActionListener#
     * actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(final ActionEvent e) {
        optimize(false);
    }
}

/**
 * .
 * @param flag .
 */
private void optimize(final boolean flag) {
    StateChart st = kielComponents[current].getCurrentStateChart();
    if (st == null) {
        return;
    }
    OptimizerChooser.startOptimize(st, flag);
    Layouter layouter = Handler.instance().getDefaultLayouter();
    try {
      layouter.setStatechart(st);
      layouter.layoutView(layouter.getStaticView());
    } catch (LayoutException ex) {
      handleException(ex.getMessage(), false, ex);
    }
  kielComponents[current].setCurrentStateChart(
      st, layouter.getStaticView());
}

/**
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
 a>
 */
class ActionResetMessages extends KielFrameAction {

    /**
     *
     */
    ActionResetMessages() {
        super("About16.gif");
    }

    /**
     * @see java.awt.event.ActionListener#
     * actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(final ActionEvent e) {
        suppressedMessages.clear();
        try {
            suppressedMessages.store(new FileOutputStream(
                new File(SUPPRESSED_MESSAGES_FILE_NAME)), null);
        } catch (IOException ex) {
            if (!(ex instanceof FileNotFoundException)) {
                handleException(ex.getMessage(), true, ex);
            }
        }
    }
}

/**
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
 a>
 */
class ActionFileExit extends KielFrameAction {

    /**
     *
     */
    ActionFileExit() {
        super("Stop16.gif");
    }

    /**
     * @see java.awt.event.ActionListener#
     * actionPerformed(java.awt.event.ActionEvent)
     */
    public void actionPerformed(final ActionEvent e) {
        boolean yes = ask(ResourceBundle.getString("exitApplication"));
        if (yes) {
            MatlabProcessInfo.shutdownMatlab();
            System.exit(0);
        }
    }
}

/**
 * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
 a>
 */
class ActionFileOpen extends KielFrameAction {

    /**
     *
     */
    ActionFileOpen() {
        super("Open16.gif");
    }

    /**
     * @see java.awt.event.ActionListener#
     * actionPerformed(java.awt.event.ActionEvent)
     */
```

```java
            public void actionPerformed(final ActionEvent e) {
                fileModel.showOpenFileDialog();
                setLoadedStatechart();
            }
        }

        /**
         * @see kiel.util.main.IKielFrame#open(java.io.File)
         */
        public void open(final File file) {
            try {
                fileModel.readStateChartFile(file);
                setLoadedStatechart();
            } catch (FileInterfaceException e) {
                handleException(e.getMessage(), false, e);
            }
        }

        /**
         *
         */
        private void setLoadedStatechart() {
            try {
              if (fileModel.getStateChart() != null) {
                if (fileModel.getActualView() == null) {
                  Layouter layouter = Handler.instance().getDefaultLayouter();
                  try {
                    layouter.setStatechart(fileModel.getStateChart());
                    layouter.layoutView(layouter.getStaticView());
                  } catch (LayoutException e) {
                    handleException(e.getMessage(), false, e);
                  }
                  fileModel.setActualView(layouter.getStaticView());
                }
                kielComponents[current].setCurrentStateChart(
                    fileModel.getStateChart(),
                    fileModel.getActualView());
                setCurrentFile(fileModel.getStatechartFile());
              } else {
                handleException(ResourceBundle.getString("fileNotFound"),
                                true, null);
              }
            } catch (FileInterfaceException ex) {
              handleException(ex.toString(),
                              false, ex);
            }
        }

        /**
         * @author <a href="mailto:flu@informatik.uni-kiel.de">Florian Luepke </
         a>
         */
        private abstract static class ExtendedKielComponent
            implements KielComponent, Preferences.Listener {

        /**
         * @return .
         */
        abstract boolean isStatusPanelVisible();

        /**
         * @return .
         */
        abstract JComponent getCompleteComponent();

        /**
         * @return .
         */
        abstract JPopupMenu getPopupMenu();

        /**
         *
         */
        public abstract void preferencesChanged();

        /**
         *
         */
        private JPanel northPanel = new JPanel(new BorderLayout());

        /**
         * @return .
         */
        protected JPanel getNorthPanel() {
            return northPanel;
        }

        /**
         *
         */
        private JPanel filler1 = new JPanel();

        /**
         * @return .
         */
        protected JPanel getFiller1() {
            return filler1;
        }

        /**
         *
         */
        private JPanel filler2 = new JPanel();

        /**
         * @return .
         */
        protected JPanel getFiller2() {
```

```
            return filler2;
        }

        /**
         *
         */
        private JPanel filler3 = new JPanel();

        /**
         * @return .
         */
        protected JPanel getFiller3() {
            return filler3;
        }
    }

    /**
     * @param isVisible .
     */
    private void setStatusPanelVisible(final boolean isVisible) {
        if (!isVisible) {
            getContentPane().remove(statusPanel);
        } else {
            getContentPane().add(statusPanel, "South");
        }
        getContentPane().invalidate();
        setVisible(true);
    }

    /**
     * @param kielComponent .
     * @return .
     */
    private ExtendedKielComponent createExtendedKielComponent(
        final KielComponent kielComponent) {

        return new ExtendedKielComponent() {

            private ToolBar[] toolBars = kielComponent.getToolBars();
            private JPopupMenu popupMenu = new JPopupMenu();
            private JComponent mainComponent = kielComponent.
 getMainComponent();
            private JComponent treePanel = kielComponent.getTreePanel();
            private JTabbedPane subPanels = kielComponent.getSubPanels();
            private JComponent statusLineComponent =
                kielComponent.getStatusLineComponent();
            private KielComponent originalKielComponent = kielComponent;
            private boolean isStatusPanelVisible = true;
            public boolean isStatusPanelVisible() {
                return isStatusPanelVisible;
            }
            private JComponent completeComponent;

            {
                Preferences.addListener(this);

                JMenu[] rootMenus = kielComponent.getRootMenus();
                // the root Menus...
                for (int i = 0; rootMenus != null && i < rootMenus.length;
                    i++) {
                    rootMenus[i].setOpaque(false);
                }

                // set the toolbars...
                final int buttonHeight = 28;

                for (int toolBarIndex = 0; toolBarIndex < toolBars.length;
                    toolBarIndex++) {
                    final int toolBarIndexx = toolBarIndex;
                    // set the toolbar components height and opaque property
...
                    Component[] components =
                        toolBars[toolBarIndex].getJToolBar().getComponents()
;
                    for (int j = 0; j < components.length; j++) {
                        ((JComponent) components[j]).setPreferredSize(
                                new Dimension(
                                    components[j].getPreferredSize().width,
                                    buttonHeight));
                        ((JComponent) components[j]).setOpaque(false);
                    }

                    // Check if the toolbars are Utils.createJToolBar()-like
...
                    if (toolBars[toolBarIndex].getJToolBar().getClass()
                        == JToolBar.class) {
                        JToolBar newToolBar = Utils.createJToolBar();
                        for (int j = 0; j < components.length; j++) {
                            newToolBar.add(components[j]);
                        }
                        toolBars[toolBarIndex].setJToolBar(newToolBar);
                    }
                    // Add item to the toolbarPanel popup-Menu for
setVisible
                    // of the toolbar
                    final JCheckBoxMenuItem item =
                        new JCheckBoxMenuItem(
                            toolBars[toolBarIndex].getMenuName(),
                            toolBars[toolBarIndex].isVisible());
                    item.setAction(
                        new AbstractAction(
                            toolBars[toolBarIndex].getMenuName()) {
                        public void actionPerformed(final ActionEvent e) {
                            if (item.isSelected()) {
                                toolbarPanel.add(
                                    toolBars[toolBarIndexx].getJToolBar());
                            } else {
                                toolbarPanel.remove(
```

```
                                toolBars[toolBarIndexx].getJToolBar());
                        }
                        resizeToolbarPanel();
                        toolBars[toolBarIndexx].setVisible(
                            item.isSelected());
                    }
                });
                popupMenu.add(item);
            }
            final JCheckBoxMenuItem statusPanelItem =
                new JCheckBoxMenuItem("Status Bar", true);
            statusPanelItem.setAction(new AbstractAction("Status Bar") {
                public void actionPerformed(final ActionEvent e) {
                    isStatusPanelVisible = statusPanelItem.isSelected();
                    setStatusPanelVisible(isStatusPanelVisible);
                }
            });
            popupMenu.add(statusPanelItem);
/*            if (getTreePanel() != null) {
                final JCheckBoxMenuItem treePanelItem =
                    new JCheckBoxMenuItem("Tree View", true);
                treePanelItem.setAction(new AbstractAction("Tree View")
    {
                    public void actionPerformed(ActionEvent e) {
                    }
                });
                popupMenu.add(treePanelItem);
            }
            if (getSubPanels() != null) {
                final JCheckBoxMenuItem informationPanelItem =
                    new JCheckBoxMenuItem("Information Panels", true);
                informationPanelItem.setAction(
                    new AbstractAction("Information Panels") {
                    public void actionPerformed(ActionEvent e) {
                    }
                });
                popupMenu.add(informationPanelItem);
            }
*/
            // set the rest...
            JComponent subPanel = subPanels;

            final int labelHeight = 18;
            if (subPanels != null) {
                if (subPanels.getTabCount() == 1) {
                    JLabel label = Utils.createJLabelHorizontal();
                    label.setText(" " + subPanels.getTitleAt(0));
                    label.setPreferredSize(new Dimension(0, labelHeight)
    );
                    JPanel p = new JPanel(new BorderLayout());
                    p.add(label, "North");
                    p.add(subPanels.getComponentAt(0), "Center");
                    subPanel =
                        Utils.createFloatableComponentForSplitPane(p);
                } else if (subPanels.getTabCount() > 1) {
                    for (int i = 0; i < subPanels.getTabCount(); i++) {
                        String title = subPanels.getTitleAt(i);
                        JComponent c =
                            (JComponent) subPanels.getComponentAt(i);
                        subPanels.removeTabAt(i);
                        subPanels.insertTab(title, null,
                            Utils.createFloatableComponentForTabbedPane(
c),
                            null, i);
                    }
                    subPanel = Utils.
createFloatableComponentForSplitPane(
                        subPanels);
                }
            }
            final int dividerSize = 3;
            final int dividerLocation = 150;
            final double dividerLocationFloat = 0.8;
            final int initialDividerLocation = 1000;
            if (getTreePanel() == null && getSubPanels() == null) {
                completeComponent = getMainComponent();
            } else if (getTreePanel() != null && getSubPanels() == null)
 {
                completeComponent = new JSplitPane(
                    JSplitPane.HORIZONTAL_SPLIT, createTreePanel(),
                    getMainComponent());
                ((JSplitPane) completeComponent).setDividerSize(
                    dividerSize);
                ((JSplitPane) completeComponent).setDividerLocation(
                    dividerLocation);
                ((JSplitPane) completeComponent).setBorder(null);
            } else if (getTreePanel() == null && getSubPanels() != null)
 {
                completeComponent = new JSplitPane(
                    JSplitPane.VERTICAL_SPLIT,
                    getMainComponent(), subPanel);
                ((JSplitPane) completeComponent).setDividerSize(
                    dividerSize);
                ((JSplitPane) completeComponent).setDividerLocation(
                    dividerLocationFloat);
                ((JSplitPane) completeComponent).setBorder(null);
            } else {
                JSplitPane top = new JSplitPane(
                    JSplitPane.HORIZONTAL_SPLIT,
                    createTreePanel(), getMainComponent());
                top.setSize(initialDividerLocation,
                    initialDividerLocation);
                // For initial divider location
                ((JSplitPane) top).setDividerSize(dividerSize);
                ((JSplitPane) top).setDividerLocation(dividerLocation);
                ((JSplitPane) top).setBorder(null);

                completeComponent = new JSplitPane(
```

```java
                JSplitPane.VERTICAL_SPLIT, top, subPanel);
            ((JSplitPane) completeComponent).setDividerSize(
                dividerSize);
            ((JSplitPane) completeComponent).setDividerLocation(
                dividerLocationFloat);
            ((JSplitPane) completeComponent).setResizeWeight(1.0);
            ((JSplitPane) completeComponent).setBorder(null);
        }
    }

    public void preferencesChanged() {
        getNorthPanel().setBackground(Preferences.getGuiColorLight()
);
        getFiller1().setBackground(Preferences.getGuiColorLight());
        getFiller2().setBackground(Preferences.getGuiColorLight());
        getFiller3().setBackground(Preferences.getGuiColorLight());
        if (getTreePanel() != null) {
            getTreePanel().setBorder(
                new LineBorder(Preferences.getGuiColorDark()));
        }
    }

    private JComponent createTreePanel() {
        final int space = 46;
        final int labelHeight = 18;
        final int spaceNorth = 6;
        final int spaceWest = 6;
        final int spaceEast = 22;
        JPanel panel = new JPanel(new BorderLayout());
        getNorthPanel().setPreferredSize(new Dimension(space, space
);

        JLabel label = Utils.createJLabelHorizontal();
        label.setText(" " + ResourceBundle.getString("treeView"));
        label.setPreferredSize(new Dimension(0, labelHeight));
        getNorthPanel().add(label, "North");

        label = new JLabel();
        label.setPreferredSize(new Dimension(0, labelHeight));
        label.setOpaque(false);
        getNorthPanel().add(label, "Center");

        panel.add(getNorthPanel(), "North");
        getFiller1().setPreferredSize(new Dimension(
            spaceNorth, spaceNorth));
        panel.add(getFiller1(), "West");
        getFiller2().setPreferredSize(new Dimension(
            spaceWest, spaceWest));
        panel.add(getFiller2(), "East");
        getFiller3().setPreferredSize(new Dimension(
            spaceEast, spaceEast));
        panel.add(getFiller3(), "South");
        panel.add(getTreePanel(), "Center");
        preferencesChanged();
        return Utils.createFloatableComponentForSplitPane(panel);
    }
    public JComponent getCompleteComponent() {
        return completeComponent;
    }
    public ToolBar[] getToolBars() {
        return toolBars;
    }

    public JComponent getMainComponent() {
        return mainComponent;
    }
    public JComponent getTreePanel() {
        return treePanel;
    }
    public JTabbedPane getSubPanels() {
        return subPanels;
    }
    public JMenuItem[] getFileMenuItems1() {
        return kielComponent.getFileMenuItems1();
    }
    public JMenuItem[] getFileMenuItems2() {
        return kielComponent.getFileMenuItems2();
    }
    public JMenu[] getRootMenus() {
        return kielComponent.getRootMenus();
    }
    public JComponent getStatusLineComponent() {
        return statusLineComponent;
    }
    public View getCurrentView() {
        return originalKielComponent.getCurrentView();
    }
    public StateChart getCurrentStateChart() {
        return originalKielComponent.getCurrentStateChart();
    }
    public void setKielFrame(final IKielFrame kielFrame) {
        originalKielComponent.setKielFrame(kielFrame);
    }
    public ImageIcon getImageIcon() {
        return originalKielComponent.getImageIcon();
    }
    public void setCurrentStateChart(final StateChart stateChart,
        final View view) {
        originalKielComponent.setCurrentStateChart(stateChart, view)
;
    }
    public String getKielFrameTitle() {
        return originalKielComponent.getKielFrameTitle();
    }
    public JPopupMenu getPopupMenu() {
        return popupMenu;
    }
```

```
}
};
}
```